Channel-and-Circuits Aware, Energy-Efficient Coding

for High Speed Links

by

Maxine Lee

S.B. EE, M.I.T., 2005

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

September, 2006

Author.....
Department of Electrical Engineering and Computer Science
September 8, 2006

Certified by.....
Vladimir Stojanovic
Assistant Professor of Electrical Engineering
M.I.T. Thesis Supervisor

Accepted by.....
Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Theses

Channel-and-Circuits Aware, Energy-Efficient Coding for High Speed Links
by
Maxine Lee

Submitted to the
Department of Electrical Engineering and Computer Science

September 8, 2006

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

Abstract

Throughput and energy-efficiency of high-speed chip-to-chip interconnects present critical bottlenecks in a whole range of important applications, from processor-memory interfaces, to network routers. These links currently rely solely on complex equalization techniques to maintain the bit error rate lower than 10^{-15}. While applicable to data rates up to 10 Gb/s on most links, this approach does not scale well to higher data rates or better energy-efficiency. The work described in the thesis shows that it may be possible to use coding techniques to share the burden of combating errors, while increasing the throughput of the link or improving its energy-efficiency. Since codes here attempt to alleviate the impact of partially correlated sources of error (like reflections interference, crosstalk and jitter), an experimental setup was created for characterization of link channel properties and performance gains from different codes. Four codes, specifically Hamming, BCH, Fire, and SEC-DED codes, are implemented and analyzed with various configurations (i.e. different blocksizes, data rates, and detection or correction). Most significantly, it is discovered that detection and retransmission of even the simple codes implemented in this project may be able to maintain a bit error rate of 10^{-15}.

Thesis Supervisor : Vladimir Stojanovic
Title : Assistant Professor of Electrical Engineering

Acknowledgments

First and foremost, thank you to my research advisor, Vladimir Stojanovic, for his continued support throughout my graduate career, and whose technical prowess and dedication to the field are truly remarkable and inspiring.

To my colleagues in the Integrated Systems Group, Natasa Blitvic, Byungsub Kim, Ranko Sredojevic, Sanquan Song, and Fred Chen, thanks for making our lab such an enjoyable environment. I could not have asked for more intelligent, fun, and fascinating people to drink coffee with and to spend all those long days and nights with.

To my best friends at MIT, especially David Vincent, my greatest supporter, and Wendy Chang, my coffee buddy and roommate, thanks for making my entire college experience amazing.

Finally, thank you to my parents for getting me started on the right track and letting me find my way. I would not be where I am today without the values and morals you ingrained in me and all of your advice throughout the years.

Contents

List of Figures

List of Tables

1 Introduction

There is a constant struggle to keep up with the increasing demands in various applications requiring high-speed chip-to-chip interconnects, such as network routers and processor-memory interfaces. To increase data rates while maintaining reliability, new advances must be made in the techniques used for data transmissions through backplanes. Backplanes are used in router backbones, where multiple, densely compact channels are needed, each transferring data at very high speeds. The purpose of the work described in this document is to demonstrate the potential of a new approach to transceiver design: energy-efficient coding. This technique should help alleviate the current bottleneck in achieving higher data rates.

1.1 The Problem

A typical high-speed link channel consists of the components in Figure 1. This system is highly complex, due to various sources of impedances, short traces like stubs and vias which are susceptible to reflections, and longer traces which suffer from attenuation. Added on top of the channel effects are the wide range of noise sources that are non-additive white Gaussian (non-AWG), such as sampling offset, supply noise, and jitter [2]. The current technique for transceiver design involves opening the eye of the signal with complex transmit and receive equalizers. However, the complexity of the equalizer and the number of taps is severely limited by the high throughput and high energy-efficiency needed by the application. Thus, to maintain high reliability, data rate is sacrificed.

An example of such a transceiver implementation is shown in Figure 2 and is described further in [3].

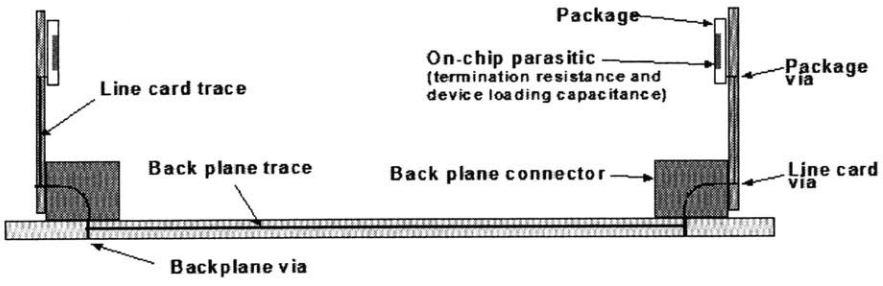

Figure 1: High-Speed Link Channel Diagram [1]

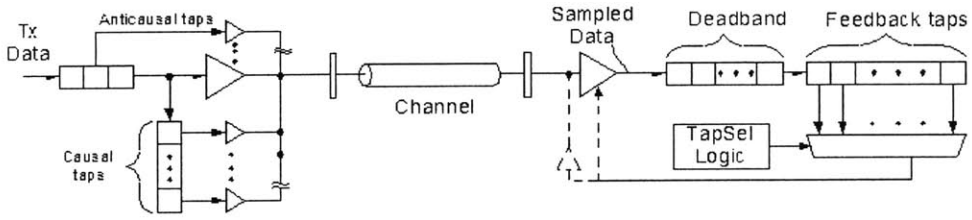

Figure 2: A Typical State-of-the-Art Transceiver Cell [3]

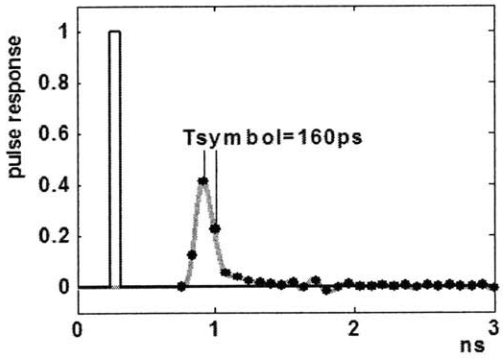

Figure 3: A Received Pulse After Link Effects [4]

The purpose of the hardware is to apply signal processing techniques to cancel the inter-symbol interference (ISI), which is deterministic. There are two types of ISI caused by the channel. The first is dispersion ISI, illustrated by the widening of the pulse in Figure 3. The effects of dispersion are short, so only short-length filters are necessary to combat the errors. The signal, however, has smaller pulses a few nanoseconds after the initial pulse because of the second type of ISI, reflections. Reflections are caused by impedance mismatches between the various connectors of the system components in Figure 1. In order to combat these reflections, current methods dissipate a large amount of power.

To combat the ISI described above and pictured in Figure 3, the hardware must be very complex and powerful. A linear equalizer called a precoder is used on the transmitting side. At the most fundamental level, the precoder consists of a simple FIR filter. The taps are designed to reduce the effect of surrounding data bits on the current data point of interest. In this particular transceiver design, five equalizer taps are used. On the receiving end, a decision-feedback equalizer (DFE) is the standard equalizer architecture. For causal ISI, the DFE can subtract away errors from the bit entering the receiver. In the design, the receiver may use up to 17 taps to cancel a single bit.

At a bit error rate of 10^{-15}, all of the required hardware burns a total of 40 mW/(Gb/s) and is only capable of operation at 10 Gb/s for 4-PAM signaling and 5-6.4 Gb/s for 2-PAM signaling. From the calculations made in [5], links should be capable of supporting data rates between 80 Gb/s and 110 Gb/s. The performance of the transceiver in [3] clearly falls short of this mark.

Finally, to illustrate the necessity of improved transceiver design, consider the result of using this state-of-the-art transceiver to create a 40 Tb/s crossbar router chip. Typical chips currently support 1 Tb/s, so 40 Tb/s is not an unrealistic increase. 4000 of the current 10 Gb/s transceivers would have to be used in the router chip to achieve the desired data rate. Since each transceiver uses a differential pair, this means that the chip would need 8000 I/O pins, requiring the total on-chip area to be 4000 mm^2. The resulting switch card would be 160 inches wide and 100 inches long. Furthermore, since each transceiver dissipates 40 mW/Gb/s, the power consumption for the crossbar chip would total 1.6 kW in 130 um CMOS technology [3].

Clearly, these results suggest that a completely new technique may be required in order to approach the theoretical capacity of links, and thus be able to create a 40 Tb/s router chip. At the very least, an order of magnitude increase in the energy-efficiency and the data rate of each transceiver is necessary. The ongoing research in high-speed links attempts to push the data rates to approach the full capacity of the link channel. For example, the advantageous parallelism of multi-tone techniques presented in [6] improves the achievable data rates significantly, but the results fall well short of the desired mark. Regardless of the modulation or equalization technique used, it is necessary to apply coding to approach full link capacity. The challenge is in doing so in an energy-efficient manner.

1.2 The Proposed New Technique

The proposed remedy to the current limitations in data rate is to add coding techniques. More specifically, reflections may be considered a noise source and handled by the code rather than the equalizers. Reflections are only weakly correlated, so coding may be much more effective than signal processing techniques at elimination their effects. The equalizer's primary job would then be to combat dispersion ISI, which requires fewer taps. Thus, the complexity of the equalizers may be reduced as well as the amount

of power dissipated. From here on, this thesis will refer to reflections as a noise source and not as ISI.

Another source of errors in link transmission not mentioned above is the signal interference caused by crosstalk. Because the signal paths in backplanes are so densely compact, the data of surrounding wires will affect the signal and possibly cause errors. Since crosstalk is not correlated with the data of interest, coding can also potentially address any issues caused by this source.

Until recently, coding for links has not seriously been considered, because the overhead required by coding and the added receiver latency appeared too difficult to surmount. Thus, it is the purpose of this work to show that there is definite potential in applying coding techniques. Existing codes are used to show that even codes not designed specifically for links can be advantageous if applied in the correct manner. Furthermore, these existing codes provide hints as to what properties are particularly effective at tackling the non-AWG noise sources of links.

The results of this work will feed directly into future research, especially involving the design of codes created specifically to address link error characteristics and the design of transceivers that incorporate both equalization and coding.

2 Background

For the remainder of the document, it is necessary for the reader to have a basic knowledge of coding theory and the particular codes chosen for implementation in the project. Furthermore, much of the terminology used in the document is defined in the following subsections. The descriptions, however, are not mathematically thorough and are only intended to provide the reader with enough insight to understand the work involved in the project. For more in depth information on coding, especially concerning mathematical concepts and proofs, see [7].

2.1 Basic Coding Theory

The purpose of coding is to increase the reliability of data transmission through a noisy medium. Some redundancy is added to the information in order to detect and possibly correct errors that occur through the channel. The typical metric for quantifying the reliability is the *bit error rate (BER)*, defined by the

total bit errors that occur divided by the total bits sent through the channel. Thus, the objective of the code is to reduce the BER.

There are two main classes of codes: block codes and convolutional codes. Block codes segment the binary sequence of information bits into k-bit blocks and calculate the redundancy for each block separately. The k bits are each transformed into n bits, where $n > k$ and $m = n - k$ is the number of redundant bits. Convolutional codes also transmit n bits for every k bits, but the parity bits are generated serially. Convolutional codes typically have very high overhead, and are not well suited for binary modulation on band-limited channels. Therefore, this thesis is focused on relatively high-rate block codes.

2.1.1 Linear Block Codes

An *(n, k) block code* is a code that transforms a *message* of k bits into a *codeword* of n bits. The ratio $R = k/n$ is the *code rate* of the block code, and it is the fraction of the bits sent through the channel that carries actual information. A block code is termed *linear* if the sum of two codewords is another codeword, and the class of *linear block codes* is a large and powerful class of codes used in this project.

The *Hamming distance*, d_{min}, is an important metric for describing the resiliency of a code. The Hamming distance is the minimum number of errors that must occur for any one codeword to be transformed into any other codeword. Therefore, a code with a Hamming distance of three will be able to detect all single-bit errors and double-bit errors within the n-bit block, since the resulting block is not a codeword. For error correction, a t-error-correcting code must have a Hamming distance of $2t + 1 \leq d_{min} \leq 2t + 2$. For example, a code with $d_{min} = 3$ can correct all single-bit errors but not all two-bit errors, since a two-bit error may cause the block to more closely resemble a different codeword.

Generator and Parity-Check Matrices A code may be defined by a $k \times n$ generator matrix, $\mathbf{G}$, or an $m \times n$ parity-check matrix, $\mathbf{H}$, where $m = n - k$. If the k-bit vector, $\mathbf{u}$, represents the message sequence, $\mathbf{v} = \mathbf{u} \cdot \mathbf{G}$ gives the corresponding n-bit codeword vector. On the decoding side, the n-bit vector, $\mathbf{r} = \mathbf{v} + \mathbf{e}$ is received, where $\mathbf{e}$ is an *error vector* that contains a 1 at a position if an error occurred. Afterward, $s = \mathbf{r} \cdot \mathbf{H^T}$ is computed, where s is called the *syndrome* of the vector. $\mathbf{H}$ is in the null space of

$\mathbf{G}$, so the syndrome will be zero only if $\mathbf{r}$ is a codeword. For $\mathbf{r}$ to be a codeword, $\mathbf{e}$ must be a codeword. The all-zero error vector, which indicates that no errors took place, is itself a codeword. Therefore, we must assume that an all-zero syndrome indicates that no errors took place. Using this approach, an error will go undetected only if $\mathbf{e}$ is a non-zero codeword, and all errors for which $\mathbf{e}$ is not a codeword can be detected. (Note: For this reasoning, it is easy to see that higher-weight codewords are desirable, where the *weight* is the number of 1's in the codeword, so that the probability of an undetected error is lower. In fact, the Hamming distance is equal to the weight of the lowest-weight codeword)

There are 2^m syndrome values for any given code, so if error correction is required by the system, then at most 2^m different error patterns, $\mathbf{e}$, may be corrected. There are, however, $2^n = 2^m \times 2^k$ possible error patterns. Therefore, there are 2^k error patterns that will cause the same syndrome value, and to maintain a high code rate, 2^k is usually much greater than 2^m. The decoder does not have enough information to distinguish between these 2^k error patterns. The decoder essentially assumes that, given a non-zero syndrome, the error pattern with highest probability occurred, and it corrects $\mathbf{r}$ accordingly. If in fact a less probable error pattern caused the syndrome value, a correction error will occur.

A code is in *systematic form* if the codeword consists of the k information bits followed by the m parity bits. To generate a code in systematic form, the rightmost columns of $\mathbf{G}$ should be the $k \times k$ identity matrix. The remaining, leftmost columns describe how the information bits form the m parity bits. Thus, $\mathbf{G} = [\mathbf{P}\ \mathbf{I_{k\times k}}]$. Similarly, for $\mathbf{H}$ to be in the null space of $\mathbf{G}$, the leftmost columns of $\mathbf{H}$ must then be the $m \times m$ identity matrix, and therefore $\mathbf{H} = [\mathbf{I_{m\times m}}\ \mathbf{P^T}]$.

Cyclic Codes Many important linear block codes are *cyclic*, which means that shifting any codeword produces another codeword. Cyclic codes can be described simply by a generator polynomial, $\mathbf{g}(X) = 1 + a_0X + a_2X^2 + ... + a_mX^m$, where the a_i's take on binary values. All codewords in the code are divisible by the generator polynomial, so the syndrome can be obtained by merely dividing the received vector by $\mathbf{g}(X)$.

Cyclic codes are especially useful because they can be encoded and decoded simply using linear feedback shift registers (LFSRs). The feedback connections are specified by $\mathbf{g}(X)$ (see Figure 4). The tradeoff in using this type of implementation is speed, for it takes k clock cycles at the encoder to create

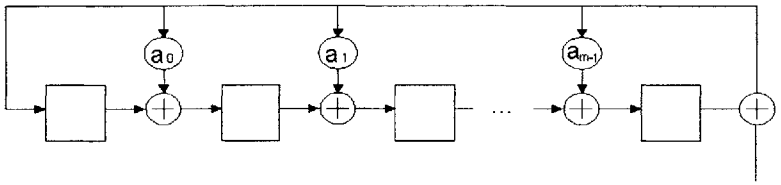

Figure 4: A simplified LFSR that generates the m parity bits for a cyclic code with $\mathbf{g}(\mathbf{X}) = 1 + a_0X + a_2X^2 + ... + a_{m-1}X^{m-1}$. a_i is a wire when $a_i = 1$. Otherwise, it is an open circuit. At the encoder, with all the registers initially set to zero, the information bits are fed into the bottom right serially. After all k information bits have been shifted in, the register contains the m parity bits.

the parity digits, and n clock cycles at the decoder to generate the syndrome digits (plus additional cycles for error correction, if necessary).

Shortened Codes Many widely used codes are specified for a particular n and k. However, different applications may require block sizes other than the one specified. It is always possible to create a *(n-l, k-l)* block code by removing l columns from the parity-check matrix, and adjusting the generator matrix accordingly. This new, shortened code has at least the error detecting and correcting capability of the original code, and often performs better at the expense of a decreased code rate.

Also, it is simple to create a *(n-l, k-l)* shortened cyclic code. A shortened cyclic codes may be generated using the same circuitry as the one used for the original cyclic code with only minor alterations in the decoder. Since shortening a cyclic code is equivalent to removing all codewords with l leading zeros, the resulting codewords are no longer cyclic. In general, a code can be shortened by deleting any columns of the parity-check matrix other than the columns that make up the $m \times m$ identity matrix. However, to keep the same hardware as the original cyclic code, the last l columns of the code's corresponding parity-check matrix must be deleted.

2.2 Types of Errors

There are two general types of errors that may occur during transmission: *burst errors* and *random errors*. Errors are considered bursty if there are multiple errors that occur within some reasonably short *length*. Thus, the following error patterns are bursty with a burst length of 6: $100001, 110001, ..., 111111$. Burst errors occur when the errors are correlated. For example, compact discs are often subject to burst errors,

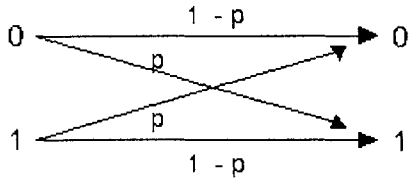

Figure 5: Binary Symmetric Channel (BSC) Model

since scratches are likely to affect multiple information bits in a row.

On the other hand, if there is no correlation between bit errors, then the errors are random. If all bit errors are independent, using the binomial distribution, it is easy to see that the probability of a higher-weight error pattern occurring is less than that of a lower-weight error pattern. Thus, for a channel with random-errors, it is beneficial to correct low-weight error patterns. The Hamming distance of a code is therefore the proper metric to judge the strength of the random-error-detecting/correcting ability of code.

Clearly, increasing the Hamming distance of a code is not the proper way to reduce the BER of a burst-error channel. Instead, burst-error-correcting and burst-error-detecting codes are designed to correct or detect a particular burst length. The Hamming distance is usually not specified in these types of codes, since the particular number of errors in the block is not the concern. Thus, a burst-error-detecting code with length seven correction ability can correct all error patterns of error length seven, but may not be able to correct a 2-bit error separated by eight bits.

It is also possible to design a code that has both random-error and burst-error detection and correction properties. Often times, this is accomplished by combining a code of a particular Hamming distance with a code of a particular burst-error-correcting length. As expected, the code suffers from a lower code rate.

2.3 The Binary Symmetric Channel (BSC) Model

A simple and commonly used channel model is the binary symmetric channel (BSC) model (Figure 5), which applies only when the probability of a bit error is independent of all other bits transmitted and the error probability is constant for all bits. In a BSC, the probability of an error occurring for any given bit is p, the BER.

It is useful to determine if the actual channel can be modeled as BSC, since it is a model based

on a single variable, the BER, which can be easily calculated experimentally. Furthermore, it would be particularly easy to set up an accurate simulation environment to test different codes. In fact, the performance of many codes has already been studied for a BSC, so choosing the right code for the application is dramatically simplified. Finally, random-error-detecting/correcting are most effective over BSC channels, so if the channel is binary symmetric, it is clear that coding will be effective.

2.4 Coding Systems

In order to capitalize on coding techniques, the codes must be implemented in some error correction protocol. In a real transmission system where the purpose is to correctly receive transmitted data, it is useless for the receiver to be able to detect an error if it is not able to utilize this knowledge to obtain the correct data. Thus, error-detecting codes must be used in conjunction with other techniques. The following sections describe techniques for error correction using codes that detect, correct, or can simultaneously do both.

2.4.1 Forward Error Correction (FEC)

A forward error correction system uses the most straightforward strategy. FEC systems utilize only the error correcting capability of codes. The transmitter sends a codeword through the channel, and the receiver attempts to correct any errors that occurred during transmission. The output of the decoder is then sent to the destination.

2.4.2 Automatic Repeat Request (ARQ)

An automatic repeat request decoder does not attempt to correct errors that occur in transmission, but instead asks for the data to be retransmitted until the block is correctly received. Thus, ARQ protocols only rely on error detection.

Automatic repeat request systems require both a forward channel and a back-channel. The transmitted data is sent through the forward channel. Once a codeword is received and decoded, the receiver transmits either a positive acknowledgment (ACK) or a negative acknowledgment (NAK) to the transmitter through the back-channel. A NAK signals to the transmitter that an error has been detected. Once a NAK is

received, the transmitter queues the corresponding word for retransmission.

Since only error detection is required, the only way for the destination to receive incorrect information is if an undetected error occurs. ARQ systems are often preferred to FEC systems, since error detecting codes require much less hardware, and error detection is much more reliable than error correction.

There are multiple types of ARQ protocols that trade off throughput and complexity. The delay through a high speed backplane is short enough such that ARQ systems have very high throughput. A typical 20 inch backplane has a round trip delay of approximately 100 symbols. The different types of ARQ systems are not studied in this project, so the reader is referred to Chapter 22 of [7] for an introduction to the different types.

2.4.3 Hybrid Forward Error Correction / Automatic Repeat Request

As suggested by the name, hybrid FEC/ARQ schemes utilize both error correction and detection with retransmission. In general, the most common errors are corrected. The less frequent errors, which may be mis-corrected if correction were attempted, are left for retransmission. Using a hybrid scheme requires further code examination, since the object of the code is no longer to maximize either correction or detection capability, but to find a compromise between the two. With simultaneous error detection/correction, the more error patterns the code attempts to correct, the fewer error patterns it will be able to detect overall. Thus, there is a tradeoff as to how much correction should be attempted.

2.5 Known Codes

A wide range of codes with varying capabilities are used in the project. Few errors per block are expected, so these codes are chosen for their lower error correcting or detecting ability and high code rate. The codes studied are Hamming, BCH, Fire, SEC-DED, and CRC codes. Specifically, Hamming codes have high rate and are capable of correcting the most common type of errors, single-bit errors, and detecting double-errors. BCH codes extend the correcting capability by one bit error and the detecting capability by two bits, since multiple errors per word happen relatively frequently. Fire codes are burst-error-correcting codes, so they provide insight into the effectiveness of burst-error-correction in links. SEC-DED codes

are a common, high rate code designed especially for hybrid FEC-ARQ systems. Finally, CRCs are cyclic codes that are used strictly for detection in ARQ schemes. More detailed descriptions of each class of codes follows.

2.5.1 Hamming Codes

Hamming codes are a very popular class of codes used for single-error-correction. Hamming codes are desirable, because they have a very simple structure and a high code rate. They are used in forward error correcting schemes when the errors are random and the error rate is low. The same code, if desired, may also be used instead for double-error detection.

Code Structure For a given number of parity bits, m, Hamming codes have a block size of $n = 2^m - 1$ for $m > 2$. Thus, the Hamming codes with the shortest block sizes are the (7, 4), (15, 11), and (31, 26) codes.

A Hamming code has a parity-check matrix of size $m \times (2^m - 1)$. The $2^m - 1$ columns consist of all the non-zero combinations of m bits. Assuming the Hamming code is in systematic form, the first m columns consist of the $m \times m$ identity matrix. This leaves the remaining $k = n - m$ columns with all the combinations of weight-two or greater.

Hamming codes may also be cyclic. Cyclic Hamming codes are generated using a primitive polynomial of degree m.

Code Properties The Hamming distance of a Hamming code is precisely three. This can be seen by examining the parity-check matrix and knowing that the Hamming distance of a code is equal to the number of columns of $\mathbf{H}$ needed to sum to zero (see [7] for a proof). Since all $2^m - 1$ non-zero combinations of m bits are used as the columns, adding any two columns together will always give the value of another column. Thus, the modulo-2 sum of these three columns gives a zero vector.

Since the Hamming distance is three, the code is able to correct all single-bit errors. Furthermore, Hamming codes are one of a few known *perfect* codes. There are exactly n error patterns of single weight and exactly $2^m - 1 = n$ non-zero syndrome values. Thus, every syndrome value can be used for single-bit

error correction. This is especially useful for FEC, since the code essentially takes full use of all the redundancy.

The Hamming distance of three also means that the same codes can be implemented to detect all errors of weight one or two. Thus, Hamming codes can be used in an ARQ system.

2.5.2 Bose, Chaudhuri, and Hocquenghem (BCH) Codes

Bose, Chaudhuri, and Hocquenghem (BCH) codes are a class of t-error-correcting codes ([8], [9]). When $t = 1$, BCH codes reduce to Hamming codes. There are a number of different types of BCH codes, and not all of them are binary. Reed-Solomon codes, for example, are a commonly used subclass of BCH codes that are nonbinary. Due to the complexity, and therefore lower energy-efficiency, involved with non-binary BCH codes, the only codes of interest in the project are binary, primitive BCH codes.

Like Hamming codes, the block size of BCH codes are $2^m - 1$. The number of parity-check digits required, however, is at most mt, where t is the number of bit errors the code can correct. For most block sizes of interest, the number of parity bits required is exactly mt.

Further discussion on the structure and properties of BCH codes is omitted, since BCH codes are constructed using mathematical tools in fields greater than $GF(2)$ (i.e. non-binary). For more information on BCH codes, the reader is referred to [7].

2.5.3 Fire Codes

Fire codes are the first class of systematically constructed single-burst-error correcting codes. Fire codes also have good burst-detecting cap

Code Structure A Fire code with a burst-error-correction length of l is generated by $\mathbf{g}(X) = (X^c + 1)\mathbf{p}(X)$, where $c = 2l - 1$. The polynomial, $\mathbf{p}(X)$ is an irreducible polynomial of degree m, where $l \leq m$. Define ρ to be the smallest integer such that $\mathbf{p}(X)$ divides $X^\rho + 1$. Then c cannot be divisible by ρ, and $n = LCM(c, \rho)$.

Code Properties To generalize the definition given above, a Fire code is actually able to simultaneously correct bursts of length l and detect bursts of length d for $d > l$ according to the equation, $c \geq l + d - 1$. Thus, the definition given in the previous section is the special case for obtaining the Fire code with greatest burst-error-correcting length. If the correction requirement is relaxed, the detecting capability of the code expands. On the far extreme, if only detection is used for an ARQ protocol, a Fire code is actually capable of detecting all bursts of length $c + 1$.

2.5.4 Single-Error-Correcting, Double-Error-Detecting (SEC-DED) Codes

The class of codes for single error correction and double error detection (SEC-DED) was first proposed by Hamming in [10] and later refined by Hsiao [11]. They are suited particularly well for hybrid FEC/ARQ systems. SEC-DED codes are designed specifically for obtaining high memory reliability in computers.

Code Structure SEC-DED codes are constructed by appropriately shortening Hamming codes in such a way that the Hamming distance is increased to four. First, all even-weight columns in the parity-check matrix are removed. Afterward, to reach the required block size, odd-weight columns of largest weight are removed. These large-weight columns are removed in such a way that the final parity-check matrix has the same number of 1's in each row, or as close to the same as possible.

Code Properties The Hamming distance of four guarantees that the code can always accurately correct single errors and detect double errors. If a single error occurs, the received vector will always be closest to the actual codeword, and correction can remove the error. If two errors occur, the received vector may be most similar to multiple codewords, so the errors can only be detected.

To see that the Hamming distance is four, we note that even-weight syndromes (like the zero-vector) must result from adding an even number of distinct, odd-weight columns from the parity-check matrix together. This, combined with the fact that the Hamming distance must be at least that of the Hamming code, from which the SEC-DED code is derived, means that the Hamming distance is at least four.

Another valuable property that results from using odd-weight columns is ease in determining whether to apply correction or detection on a received vector with a non-zero syndrome. If the syndrome is

odd-weight, an odd-weight error pattern must have occurred. The decoder assumes that a single error occurred, and it attempts to correct the received vector. If the syndrome is even-weight, then at least two errors have occurred, so only retransmission is possible.

SEC-DED codes are designed for encoding and decoding the entire codeword in parallel. For this reason, the algorithm creates a parity-check matrix with a minimum number of 1's and an even distribution of 1's between the rows. These two requirements guarantee that minimum logic levels are used in calculating each syndrome bit, allowing for faster clock speeds.

2.5.5 Cyclic Redundancy Check (CRC)

A cyclic redundancy check code is simply the term used to describe a cyclic or shortened cyclic code used for error detection within an ARQ system. Thus, if a cyclic Hamming, BCH, or Fire code is used only for error detection, it is considered a CRC.

Code Structure A CRC is defined by its generator polynomial, $\mathbf{g}(X)$. There is no systematic way to choose $\mathbf{g}(X)$ and no boundary on what the block size should be given a particular $\mathbf{g}(X)$. Often times, a "standard" polynomial is chosen for an application only because it is a commonly used polynomial, and may in fact be inferior to many other polynomials.

Code Properties Because there is no defined structure for a CRC code, it is not possible to determine the Hamming distance of the code without knowing both $\mathbf{g}(X)$ and the block size. In [12], the authors provide insight into the effectiveness of commonly used generator polynomials and a few new polynomials. Furthermore, from an exhaustive search, they determine the best $\mathbf{g}(X)$ for a given information length, k, and CRC size, m.

The one property common with all CRC codes is the burst error detection capability. As with any cyclic and shortened cyclic codes, CRC codes can detect all bursts up to the CRC size.

3 Previous Work

Although coding is a commonly used technique for increasing transmission reliability, it has typically been dismissed as implausible for use in high-speed links. This is due to the fact that the throughput requirement of links is extremely high, and using a code immediately, and often drastically, reduces the information rate. Only recently has research started in the field of link coding.

In [13], a (62, 56) shortened Hamming code was used for a link running at 10.3125 Gb/s. The code was able to reduce the bit error rate by many orders of magnitude, from 3.2×10^{-9} to 3.86×10^{-16}. This work shows that there is great potential in applying coding to high speed link transmissions, since such a simple, general code was able to achieve such impressive results. More recently, a run length code combined with a double-error-correcting, primitive BCH code was demonstrated. [14]

The authors of [15] took another approach to link coding. They developed a 4-PAM coding scheme that improves performance by eliminating full-transition swings. Certain worst case patterns are completely removed, improving the timing margin and decreasing distortion. The work is quite successful at demonstrating the possibility of designing a code that is aimed specifically at combating high-speed link noise.

4 Problem Statement

4.1 Purpose of the Work

4.1.1 Limitations in Current Methods in Link Transmissions

As described in the Introduction, currently the only method of combating errors in link transmissions is to use transmit and receive equalizers. However, due to factors such as reflections, which affect several subsequent symbols, in the current design approach, the only way to increase the performance of a link is to find a new way to increase the number of equalizer taps in the power-constrained environment. Even if new circuit techniques were discovered to increase the number of taps, it does not seem likely that an entire order or magnitude increase in both energy-efficiency and data rate could be achieved.

4.1.2 The Goal of This Work

Rather than using the complicated equalization techniques described above, a system based on both equalization and coding is being developed. With this added technique, the raw BER requirement (i.e. without coding) may be reduced to as low as 10^{-6}. This will significantly reduce the complexity and hardware requirements of the equalizers, and thus reduce the power burned by the transceiver. To bring the BER back down to 10^{-15}, a carefully chosen or designed code will be implemented, which itself will have some associated hardware and burn some amount of power. The initial goal is to obtain the same reliability as current implementations at higher energy-efficiency.

The specific goal of this project is to examine the potential of link coding and determine the direction that future research in this area should take by obtaining information and providing answers to the following questions:

- *Study the effectiveness of current, commonly-used codes in links.* There are a wide range of codes being used today, and they all have very different properties. If a simple Hamming code, as described in [13], is capable of obtaining such remarkable results, a desirable question that needs to be answered is: what other codes would work better? And more significantly, what properties do these codes possess that make them so appropriate for the link environment?

- *Provide a general suggestion on which of the commonly-used codes may be used and in what situations.* Many codes behave differently in different situations, so, for example, a code that works well at a certain data rate may behave poorly at another. Also, the code rates vary quite significantly from one code to the next, so there may be block sizes that some codes just should not support. The important question to answer is, given a set of criteria on block size, data rate, etc., what is the best code to use? What makes that code the better than all of the others?

- *Rationalize the use of error-correcting codes for an FEC system, error-detecting codes for an ARQ system, or a hybrid of the two.* As presented in [16], a back-channel can be obtained by using common-mode signaling. Therefore, it is plausible for all three types of coding protocols to be used. The requirements of an ARQ system are quite different than those of an FEC protocol. It is

therefore necessary to determine which type has the most potential to work for links.

- *Give a recommendation on how to progress with the study of coding for links.* The work in this project ranges from studying random-error vs. burst-error detecting/correcting codes to studying error detecting versus error correcting codes, and even to finding the appropriate block size. With all of these variables to examine, is there a single combination that is clearly superior? If not, then what are the most promising results that should be looked into further in the future?

The first step in achieving the goal is to study the type of errors that occur in links. Pseudo-random data is sent through the link in order to obtain the error statistics of raw, uncoded data. After examining the error statistics of the link, it is then possible to make a reasonable prediction as to what properties (i.e. Hamming distance or length of burst-error correction/detection capability) an effective code should possess. Also, the error statistics will reveal what block sizes would be reasonable to implement.

Based on these findings, a few known codes are chosen with some or all of the necessary properties. The chosen codes are then implemented in an FPGA test bench for three different block sizes. An FPGA framework is used, because it offers flexibility in code design and complexity evaluation for different schemes while cutting down the time to build a custom, equivalent system.

Based on both the predicted and experimental results of the code, preliminary conclusions can be made as to what type of code is most suitable for link applications. Then suggestions for future research can be recommended.

4.1.3 Limitations in Previous Methods in Link Coding

There have been some attempts at coding for links, as described in the Previous Works section, but they do not attempt to answer any of the questions stated above. There are further limitations as well that prevent their results to be of particular use to this project.

In the first example ([13]), although an amazingly low BER is observed, it can not be attributed only to the Hamming code. Due to hardware constraints, the data is protected by both 8B/10B and 64B/66B, which essentially eliminates all the worst-case data patterns. Also, in order to increase the amount of errors, a noise source external to the system was used to inject errors into the system. This noise source

is the dominant noise source, so the results are not an accurate reflection of the performance of the code at preventing errors caused by typical link noise.

Furthermore, only one code, the shortened (62, 56) Hamming code was implemented, which was chosen only for its simplicity and small block size. The purpose of the work was to show in a very basic, preliminary test that coding for high-speed links has potential for improving the bit error rate significantly, which it was very successfully at doing. However, the work does not attempt to further our knowledge of link behavior when coding is used, nor does it broaden our understanding of how effective other codes in general can be.

The main issue with the code developed by [15] is the large overhead. The code rate is only 80%. Such a low throughput is generally undesirable, and, due to the highly low pass characteristic of links, increasing the bit error rate in links to increase the information rate will likely remove most, if not all, of the improved effects of the code.

4.2 Simultaneous Work and Future Plans for Link Coding

The work in this project is part of an extensive, multi-stage effort to develop a system that achieves high energy-efficiency and data rates that are closer to the theoretical limit. The high level stages of the project are:

- Phase 1: Link Simulation
- Phase 2: Power-Performance Analysis
- Phase 3: Design of New, Energy-Efficient Codes
- Phase 4: Implementation of New Codes

The first two stages are preliminary, setup phases for the overall project. This project is part of the second stage. The first stage is being studied and implemented concurrently. Specifically, the purpose of the Link Simulation phase is to set up an accurate and reliable simulation system to implement and test different codes. The simulation environment is especially important since the physical hardware designed

to provide a general, all-purpose solution, and therefore has natural limitations that will not be a factor once a custom test chip is built.

The Power-Performance phase will be completed once the codes studied in this project are actually implemented in a complete coding system (i.e. ARQ) using a theoretical Matlab model and, hopefully, the link simulator. At that point, the energy-efficiency of the system, which is affected by both the code and type of coding system, may be determined. This information, in addition to the performance of the model, is sufficient for obtaining a quantitative evaluation of the power-performance tradeoffs of various known codes.

The design of new codes will be directed by the results of the power-performance analysis of known codes. The design will attempt to utilize and incorporate those properties of known codes that are effective at combating link noise sources. The performance of the code may undergo preliminary tests using the same test hardware as was used in this project. The actual performance, however, will not be known until a custom circuit is designed in the final stage.

Designing a link that runs at such high speeds is a very difficult task. The new codes designed will likely have specific hardware requirements for the encoder and decoder. Thus, the circuit designer working on the fourth stage of the project must not only design a high-speed transceiver, but must also incorporate a high-speed encoder and decoder and all associated hardware, as well as any extra hardware required by the coding system.

The overall goal of the project is quite ambitious, and if the resulting test chip can be shown to work effectively, it will make a huge impact in link transmission methodology. The results of this intermediate project are very significant to the overall project, since it shows just how much potential there is in coding for high-speed links and that future study in this direction is desirable.

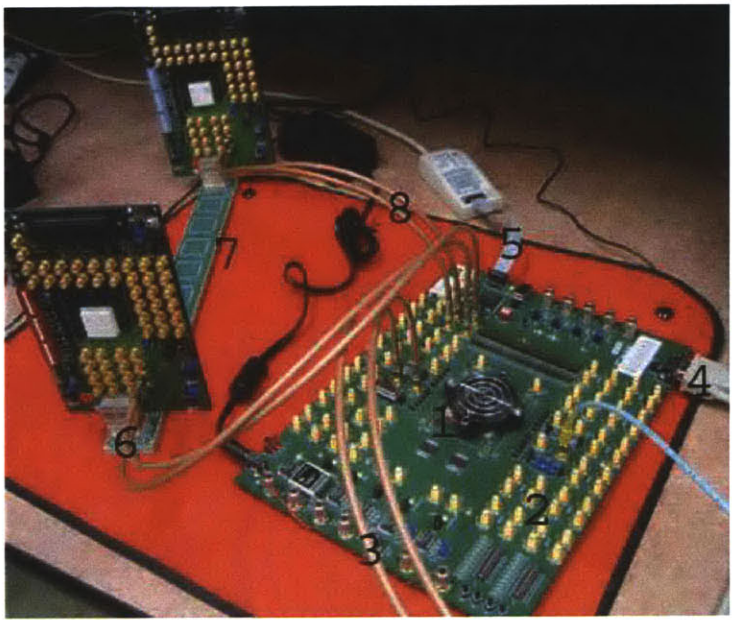

Figure 6: Laboratory Setup. The labels on the figure indicate the 1) Virtex-II Pro X FPGA, 2) RocketIO X SMA Connectors, 3) Differential clock from external signal generator for clocking the transceivers, 4) RS232, 5) JTAG Cable for programming the FPGA, 6) Differential transmit coaxial cables, 7) high-speed backplane, and 8) differential receive coaxial cables.

5 Methodology

5.1 Description of Hardware

The hardware required for the project includes a Xilinx FPGA with built-in transceivers, Rambus line cards and backplanes, coaxial cables, and other general laboratory equipment. The entire set up is shown in Figure 6. A pseudo-random data stream is generated in the FPGA, encoded, sent through the backplane and brought back to the FPGA via the coaxial cables and line cards. Once the FPGA receives the transmitted data, it can be decoded and any errors that occurred may be observed and recorded.

5.1.1 Virtex II-Pro X FPGA

The FPGA board used in the project is a general purpose demonstration board, called the MK325, designed by HiTech Global. It contains a Virtex II-Pro X FPGA.

This particular FPGA has many built-in features that make it suitable for the demands of the project, including:

- *Digital Clock Managers (DCMs):* The FPGA has eight digital clock managers that can multiply and divide the frequency and change the phase of an input clock. The DCMs allow the user to simply create a system with multiple clock domains without having to bring in extra off-chip clock sources.

- *Block RAMs:* The Virtex-II Pro X has 308 block RAMS of 18kbits each, which can be combined to make larger storage devices. These block RAMS are suitable for implementing FIFOs and RAMs, which are not efficiently implemented using logic cells.

- *RocketIO X Transceivers:* The RocketIO X transceiver is the I/O device that is used to drive the device under test (DUT). The next section is devoted to describing the RocketIO X.

- *IBM PowerPC Processor:* The FPGA has two embedded 32-bit IBM PowerPC processors. The PowerPC processors uses block RAM for its data and instruction memory. PowerPC programs are programmed in C and then loaded into the block RAM when the FPGA is programmed. The processors may be utilized if a software solution is more appropriate to parts of the design. The PowerPCs may also be effectively used for controlling drivers for FPGA peripherals, such as RocketIO drivers.

  The processors communicate with its peripherals via two buses, called the Processor Local Bus (PLB) and the On-Chip Peripheral Bus (OBP). The designer must connect the peripheral drivers to one of these buses and define its address space. The drivers can then be accessed and manipulated using a the standard address/bus interface.

5.1.2 RocketIO X Transceivers

The RocketIO X, as is typical for high-speed applications, uses differential lines for transmitting and receiving. The MK325 brings out all four pins for each of the 20 transceivers to SMA connectors for a total of 80 SMA connectors. These connectors provide the interface between the RocketIO I/Os and the channel.

The RocketIO transceivers are designed to operate up to 10 Gb/s, though because of board routing and other issues, the MK325 only guarantees operation up to 6.25 Gb/s on all its the transceivers. The

transceiver used in this project was selected based on testing all 20 transceivers for the lowest bit error rate. The selected transceiver is capable of 10 Gb/s operation for channels of relatively high quality, including the channel that consists of a single coaxial cable.

The RocketIO accepts 40 bits of parallel data and transmits the data serially at 40 times the reference clock. The bits are sent from the least significant bit to the most significant bit. For 10 Gb/s operation, the FPGA logic must be clocked at 250 MHz from an external signal generator.

The RocketIO X communicates with the FPGA fabric through dedicated registers. Other than the 40-bit transmit and receive data registers, there are many other registers that control the various transceiver settings. Many of the features of the RocketIO X are bypassed, such as 8B/10B or 64B/66B encoding, so the full effects of the channel and the coders/decoders can be examined. The RocketIO transmit and receive equalizers, however, are also controlled by registers and must be tuned for each channel and each data rate in order to eliminate excessive dispersion ISI. These more advanced features of the RocketIO X can be manipulated using a special bus, called the Physical Media Attachment (PMA) Attribute Programming Bus.

The RocketIO uses very simple equalizers that have rather coarse resolution. On the transmit side, two separate registers combine to control the output swing and pre-emphasis levels. Between the two registers, there are a total of 72 combinations. Depending on the chosen values, the output swing ranges from 120 mV to 780 mV (single ended) and the pre-emphasis levels ranges from 0% to 500%. On the receiver side, the magnitude of a highpass filter is controlled by a 10-bit register, where two bits control the filter value between 50 MHz and 200 MHz, two bits control between 200 and 500 MHz, and six bits control between 500 MHz and 2 GHz. For more information on the RocketIO X, refer to [17].

5.2 Desired Information to Capture

The purpose of the hardware system is to obtain enough information to get a reasonably clear picture of how the channel noise affects the link. Other than obtaining the bit error rate, there is a wide range of valuable information that can be captured about the error distribution. The type of errors caused by channel noise directly determines what kind of code will be most effect at correcting or detecting them.

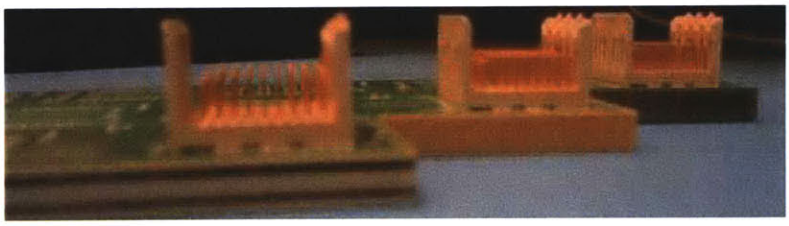

Figure 7: Different Materials Used in Backplanes. From front to back, the picture shows a Rogers, NELCO, and FR4 link.

The error distribution affects three crucial aspects of a code: the expected Hamming distance required by a code, the expected burst length correction/detection capability, and the range of block sizes, n, that can be considered.

The expected number of errors in the block increases as the block size increases, but the code rate increases as well. Thus, there is always a tradeoff between the block size and the needed detecting/correcting capability of a code. It is this tradeoff that the designed system must be able to evaluate. Even though the hardware imposes some restrictions on the block size, it is necessary that system be able to handle a range of block sizes. For each block size, obtaining the distribution of both the burst length of errors and the number of errors in the block completes the picture.

5.3 Line Cards and Backplanes

The line card used in the project is pictured in the top left corner of Figure 6. Typically, data is generated on the white chip in the middle of the line card, as is diagrammed in Figure 1. However, in this case the data is brought in externally using coaxial cables, also pictured, from the Virtex-II Pro X FPGA.

There are multiple types of backplanes available for the project, each of which are 20 inches long. Figure 7 pictures backplanes with different substrates. The figure shows, from front to back, a Rogers, NELCO, and FR4 backplane. FR4 is the typical material used for circuit boards. However, FR4 may be too lossy at the high bandwidths required of high-speed links. NELCO and Rogers are alternatives to FR4 and provide lower loss solutions.

There are many sources of reflections due to discontinuities in the signal path. One way of reducing reflections between the line card and the backplane is to drill out the vias connecting them, a technique

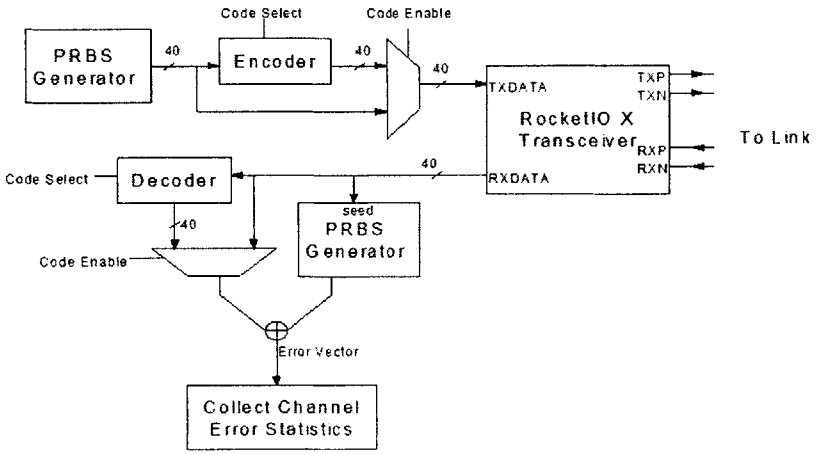

Figure 8: High Level Block Diagram

called counterboring. The backplanes used in the project do not utilize counterboring, since they are the more interesting case. Thus, due to availability, the choices are limited to NELCO and Rogers. Of the two, the Rogers link is chosen for study, because it results in more errors and is therefore more interesting.

5.4 High Level Description of the Design

In its simplest form, the system can be viewed as the diagram in Figure 8.

At one end, the transmitter generates a pseudo-random bit sequence (PRBS) in blocks of 40, codes the sequence, and sends it through the channel. The receiver then uses the transmitted data to seed a replica of the transmitter's pseudo-random number generator. As long as the seed value is error-free, the receiver's PRBS generator can produce the same sequence as the transmitter's. When the receiver can accurately predict the incoming data, a *locked* state has been achieved. Once locked, the receiver need only compare the incoming data to the generated data to determine whether any errors have occurred. If there are errors, then the error vector is saved for processing. In the processing stage, both the error length and the number of errors are calculated, as required by the discussion in the previous section.

In the next subsections, each portion of the design is discussed more thoroughly. The transmitter and receiver design is based loosely on the Xilinx RocketIO Bit Error Rate Tester application note [18].

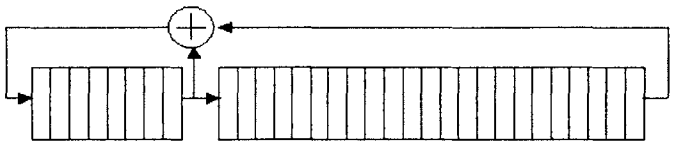

Figure 9: LFSR Implementation for the PRBS generated by $g(X) = X^{31} + X^8 + 1$.

5.4.1 PRBS Generator

The pseudo-random number sequence is generated by inverting the sequence produced by the polynomial, $X^{31} + X^8 + 1$ as recommended by the International Telecommunications Union (ITU-T) [19]. The polynomial generates a sequence of $2^{31} - 1$ bits before the entire sequence repeats again. For data rates between 5 and 10 Gb/s, the sequence repeats every 0.21 to 0.43 seconds.

Only one bit is produced in a clock cycle if the standard LFSR implementation of the polynomial is used. Unlike the LFSR shown in Figure 4, a generator polynomial for a random number generator usually indicates the taps used to create the newest value entering the most significant register. The xors for this form of LFSR are external to the shift registers, while the logic in Figure 4 is implemented between the registers. The polynomial used for the PRBS, $X^{31} + X^8 + 1$ is diagrammed in Figure 9.

Since the RocketIO requires 40 bits of data per clock cycle, the output of 40 LFSR shifts must appear at every clock cycle. The first 31 bits can be generated by creating a circuit that does the equivalent of 31 shifts in a cycle. Then all 31 registers in the LFSR will contain new data, which can be taken in parallel and sent to the encoder or to the RocketIO. The proper logic to implement the circuit can be calculated using matrix math. A single shift can be represented in vector notation by $\mathbf{v_n^T} = \mathbf{A v_{n-1}^T}$, where $\mathbf{v_i}$ is a 31-bit vector of the values of each register in the LFSR for cycle i. $\mathbf{A}$ is a 31×31 matrix with one row consisting of the generator polynomial in vector form, and 30 rows of weight one to show that each register's value depends only on the register before it. For two shifts, the equation becomes $\mathbf{v_n^T} = \mathbf{A(A v_{n-2}^T)} = \mathbf{A^2 v_{n-2}^T}$. Generally, for x shifts, $\mathbf{v_n^T} = \mathbf{A^x v_{n-x}^T}$. Thus, rather than implementing $\mathbf{A}$, like in Figure 9, the circuit must implement $\mathbf{A^{31}}$. The jth row of the matrix, A^{31}, gives the feedback taps necessary to construct the jth bit in the LFSR. This technique is discussed more thoroughly in [20].

If this circuit is extended to include 40 registers, then 40 bits of parallel data can be generated in one cycle. Notice that even though this implementation requires a 40-bit register, it only needs a seed of 31

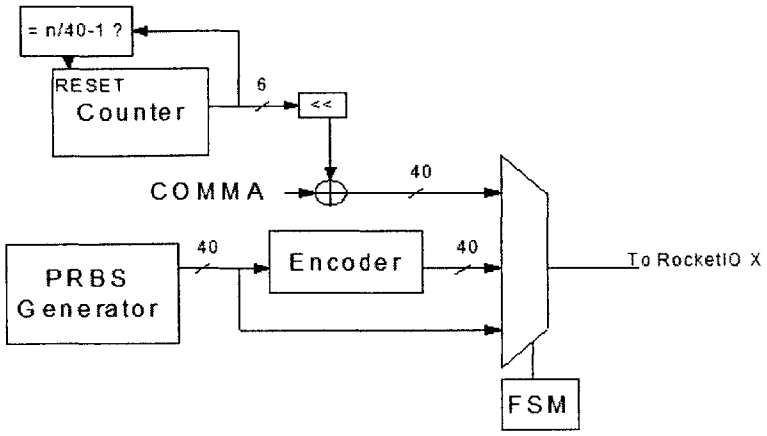

Figure 10: Diagram of the Transmitter. The FSM chooses between the three options: the comma sequence, uncoded PRBS, and coded PRBS. The counter loops indefinitely through the block numbers. The encoder must also know the current block number, but the connection is left out of the diagram for clarity.

bits.

5.4.2 Transmitter

The transmitter is the module that determines the data that is sent through the link. Overall, the functionality of the transmitter can be described by the finite state machine shown in Figure 11. Also, a more accurate diagram is shown in Figure 10.

Before the transmitter can send coded data through the channel, it must send two initialization sequences to ensure the receiver can process the data correctly. The first sequence allows the receiver to align the transmitted data on the proper 40-bit boundary. Correct alignment is necessary when data is coded, because the receiver must be able to distinguish between information and parity bits. The transmitter first sends a comma sequence, where each 40-bit word contains a special 10-bit character aligned at the beginning of the block. When the receiver detects the presence of this signal, it adjusts the data alignment such that the 10-bit character takes up the 10 least significant digits of the 40-bit word. The transmitter sends 4096 words with comma characters to guarantee that the receiver has had enough time to align the data accurately.

In a similar manner, for block sizes greater than 40, the receiver must be able to distinguish between

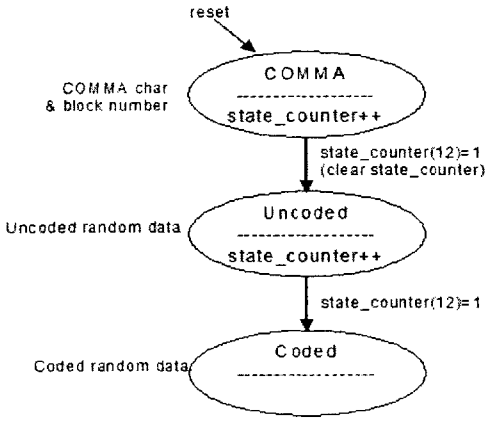

Figure 11: Transmitter FSM. The data outputs of each state are shown on the left.

the position of each 40-bit block within the codeword. Since the codeword length is not changing within a single test, this information may be communicated during the initialization sequence. The comma sequence is first divided into $n/40$ groups. Thus, each codeword also has a relative position within its group. Each word is assigned a number between zero and $n/40 - 1$ to signify its position. There are no requirements for the remaining 30 bits within a comma word, so its position is transmitted using only six of the leftover bits. When the real data sequence begins, its block positions are kept in phase with the comma sequence's positions. Given that 4096 comma words are transmitted, the receiver has an ample number of cycles to achieve codeword alignment.

After the comma sequence is transmitted, another 4096 40-bit blocks of uncoded data is sent. The purpose of the uncoded data sequence is to allow the receiver's PRBS generator to lock to the incoming data, which it might not be able to do with a coded sequence. The only situation where a problem arises is when more than nine bits in a 40-bit codeword is used as parity. As stated in the previous section, the generator requires 31 bits as a seed, so this type of coded sequence is insufficient to start up the PRBS generator.

After the initialization sequences, the transmitter is ready to send coded data. The coded data is synchronized with its uncoded counterpart and with the block number from the comma sequence. The transmitter stays in this state until the test is stopped.

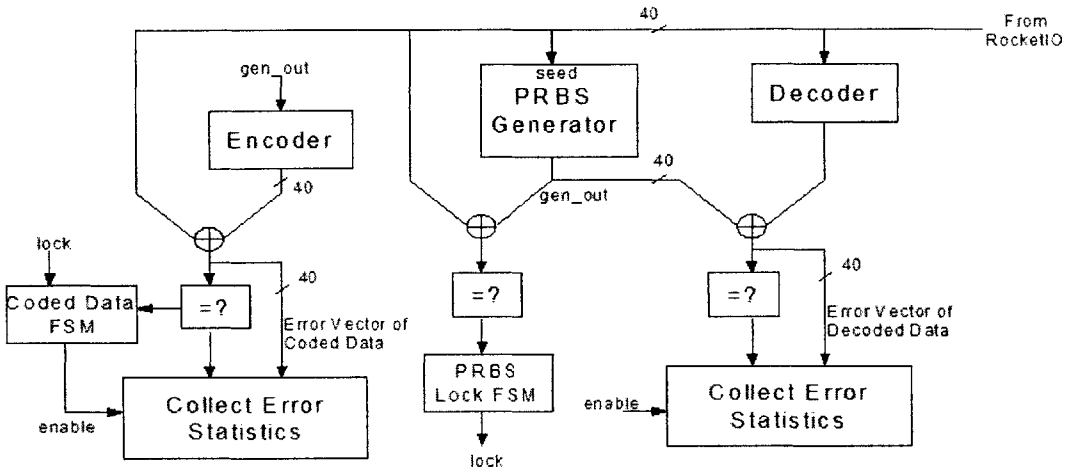

Figure 12: Receiver Architecture. All of the main components other than the alignment FSMs are shown. Also, all timing hardware is omitted for clarify.

5.4.3 Receiver and Statistics Gatherer

A more detailed version of the receiver architecture is shown in Figure 12. The receiver is made up of two alignment modules, the decoder, three error detection circuits, and two statistics accumulators. The alignment modules find the boundaries between each codeword in the serial stream. The three error detection circuits determine whether there is an error in the vectors of the uncoded data, the coded data, and the decoded codeword, respectively. The statistics accumulators save the error properties of each codeword that is received with errors.

Initialization and Alignment During the initial comma sequence, the receiver has two modules that align the 40-bit word boundaries and the codeword boundaries, respectively. The RocketIO automatically accomplishes the first task if the appropriate register is enabled. During normal operation, automatic alignment should be disabled so that no word in the random sequence that has the special 10-bit character causes the RocketIO to shift its alignment. Thus, the first module has the responsibility of determining when to enable the RocketIO automatic alignment register and the length that the function is enabled. At the beginning of the test, the function is enabled. Once the module detects 256 straight words with the 10-bit alignment character, it disables automatic realignment. The number, 256, is chosen to be significantly less than the total number of comma words to account for errors in the stream, yet it is large

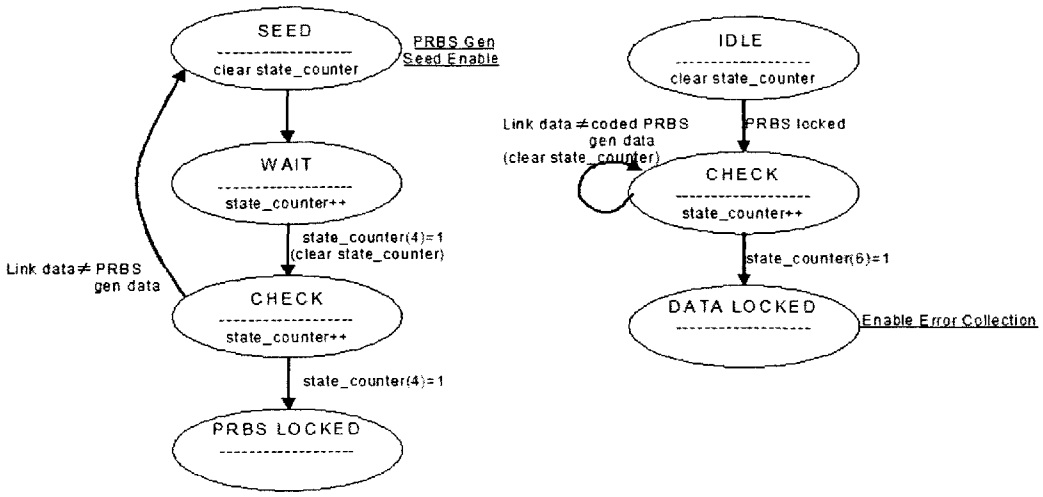

Figure 13: Receiver Finite State Machines

enough to ensure alignment.

The second module locks a counter to the incoming block numbers. Once the counter consistently matches the block numbers in the comma sequence, the counter loops independently of the incoming data, and continues to do so indefinitely. By the time the comma sequence ends and actual data is sent, the block number of each incoming word is known.

Receiver Finite State Machines (FSMs) The purpose of the receiver is to collect the error statistics of the link. However, it must wait until after the initialization and alignment stages are over so that the received data is valid. The receiver has two finite state machines that keep track of the current stage (see Figure 13).

The first FSM locks to the uncoded data sequence, which is the second initialization stage, described in the transmitter section. The FSM seeds the PRBS generator, waits a few clock cycles, and then determines whether the incoming data matches the generated expected data. Since the received data may still be part of the comma sequence or may contain errors, the receiver's PRBS may be incorrectly seeded. The receiver, therefore, must re-seed the PRBS generator as many times as is necessary to lock the output of the receiver's PRBS generator to the incoming sequence. Once the two sequences are in lock step, the first FSM enables the second.

The second FSM simply waits for the start of the coded sequence. Instead of comparing the output of the PRBS generator to the received data, it compares the incoming link data to the coded version of the generator output. Once the two data streams match for several codewords, the FSM assumes that it is now receiving the coded sequence. At this point, the received data is considered valid for error statistics collection.

Gathering Error Statistics Two error statistics gathering modules are implemented in the receiver. One accumulates data from comparing the incoming link data, which is itself coded, to the coded version of the expected data. The encoder, as shown on the left side of Figure 12, is used to code the pseudo-random bit sequence and is an exact replica of the transmitter's encoder. The statistics obtained by this comparison reveals the error distribution of the coded sequence. Because the code adds data dependency, the stream sent through the link is not random, and the error distribution may be altered. Therefore, each code must be evaluated for how well its coded data stream behaves through the channel as well as how well the actual code performs.

The other statistics gathering module collects data from comparing the decoded channel data to the uncoded, generated data. This path is shown on the right side of Figure 12. When correction is implemented and enabled, the distribution of the uncorrectable errors, including those that are mis-corrected or undetectable, is collected. When detection is enabled, the module collects all errors that are undetectable.

Comparing the gathered distribution of both modules reveals pertinent information about the code. Essentially, the two modules show the error statistics of the data before entering the decoder and the resulting data after correction or detection. This is useful for examining how many errors, and sometimes which errors actually make it past the decoder to the destination.

The modules used for collecting error statistics each need only an error vector as input. The error vector alone is sufficient for determining both the length of the error and the number of errors within the 40-bit word. The number of errors in a 40-bit word can be calculated by adding the number of ones in the vector. The length of the error can also be calculated by determining the bit positions of the first and last error in the word, and subtracting the two positions. For codewords larger than 40 bits, extra

logic is required to perform these calculations for groups of 40-bit blocks.

The error processing logic is clocked with a slower clock than the clock used to generate the error vector, so a FIFO is used to interface the two clock domains. If an error is detected, which can easily be determined by a detection circuit that checks if the error vector is non-zero, the entire error vector is saved in the FIFO. The error processing logic is clocked at a slower rate, because it is unnecessary and impractical to force these circuits to function as quickly as the detection circuit. Error vector generation must occur at the same rate that data is received from the RocketIO. For 10 Gb/s operation, this translates to a logic clock of 250 MHz. However, errors occur rather infrequently (typical bit error rates range from 10^{-5} to 10^{-10}). Thus, as long as the error processing clock pulls out error vectors from the FIFO as fast as the vectors enter it, the circuit will function properly. For the design, a 50 MHz clock is used, which is sufficient for error rates below 3×10^{-3}. The FIFO used in the design has a depth of 512 to handle bursts of errors that span multiple words.

As previously stated, it is important to collect the error length and the total number of bit errors each time a codeword is received incorrectly in order to obtain the distribution of all the errors. Nine RAMs are used to create histograms of the error data. Eight are used to histogram error lengths. Error lengths are classified based on the number of errors that occur in the codeword. For example, the first RAM stores the error length of words with two bit errors, and the eighth RAM stores the lengths of the errors when nine or more bit errors occur in the word. There are eight length histogram, because, even for very large blocksizes, more than eight bit errors per word occurs very infrequently. The ninth RAM is used to histogram the number of bit errors in a word.

The circuit that generates the histogram is shown in Figure 14. At the beginning of the test, all of the RAM's addresses are set to zero to signify that no errors have occurred. The calculations for obtaining the number of bit errors in a word or the error length is used as the address to the RAMs. The resulting calculations for each error vector appear for a single cycle, so a 100 MHz clock is used to clock the RAM, which is twice as fast the the error processing clock. This divides the slower clock cycle into a read and a write cycle. In the read cycle, the RAM address containing the error length or number of errors is read, and the number of previous occurrences of the event appears at the RAM's data output and is fed

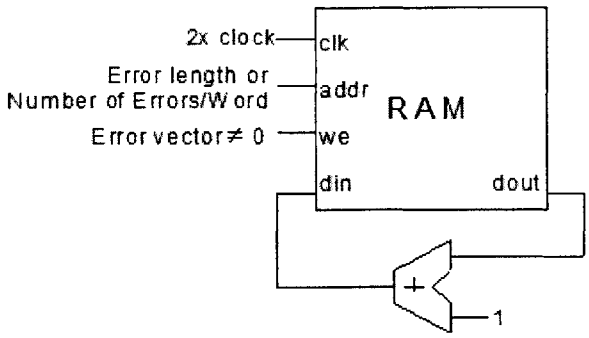

Figure 14: RAM Connections to Create a Histogram

through an incrementer. In the write cycle, the incremented value is written back into the same address. A cycle-by-cycle example of the histograming process is shown in Figure 15.

The RAMs storing the histograms are all dual-port RAMs. These connections are omitted from Figure 14. The extra connections allow an external source to read the values stored in the RAM at a different clock frequency. In the design, the PowerPC reads the results at the end of each test, displays them to the user, and resets all the addresses before the next test starts. The PowerPC system is described further in the next section.

5.4.4 Software System

The test system must have a way for the user to easily change the structure of a test, such as which code is enabled. Other than the RocketIO transceivers, the MK325 has a RS232 connector to interface the FPGA to an external system. The most straightforward way to use the serial port is to write a software program for the embedded PowerPC. The PowerPC program acts as the main controller that controls its two peripherals, the RS232 driver and the RocketIO driver described above. The processor communicates with the peripherals using the on-chip peripheral bus (OPB), which is clocked at 100 MHz.

The processor interacts with the RocketIO hardware system using several registers, some of which are read-only, write-only, or read-and-write. The address space of the peripheral is shown in the Table 1.

Using the serial port to communicate with the user, the system is able to change the test parameters between tests without having to reprogram the FPGA. Once the user finishes setting up a test, the processor communicates the settings to the hardware via the registers listed in Table 1. Some of the

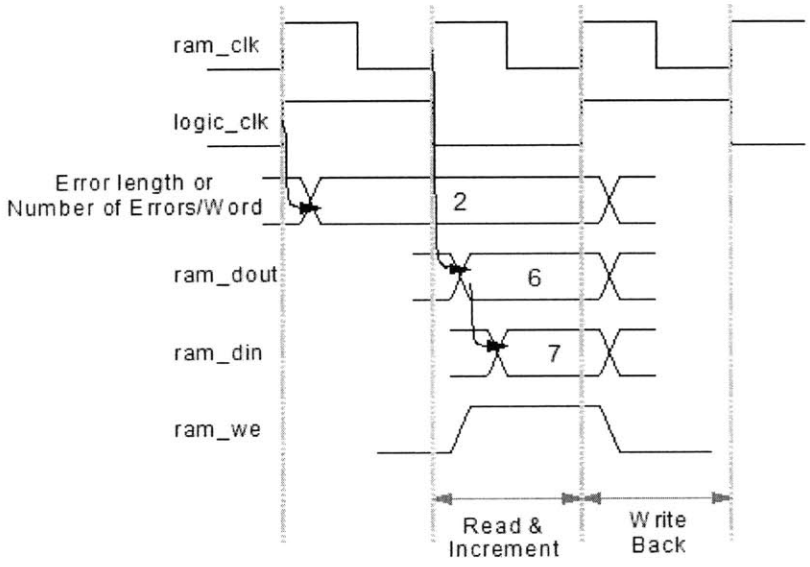

Figure 15: Cycle-by-Cycle Example of Creating a Histogram Using a RAM. In the example, an error length of two is calculated. In the read cycle, addressing the RAM with the value two, six is read out, signifying that there are six previous occurrences of length-two errors. Six gets incremented to seven as shown in Figure 14 and is written back into address two of the RAM.

addr	Write Only	Read Only
00	Reset	Get Test done, Test error
04	Set Block Size	
0C	Set Code	
10	Set Data Type	
14	Set Detection/Correction/Hybrid	Get PRBS Generator Lock
18-1C		Get Total 40-Bit Words
20-24		Get Total Bit Errors
28		Get Total Frame Errors
30	Set PMA Attributes	Get PMA Attributes
300000	Errors/Codeword RAM Base Address (for coded data)	
300FA0	Errors/Codeword RAM High Address (for coded data)	
700000	Error Length RAM Base Address (for weight-2 errors and coded data)	
700FA0	Error Length RAM High Address (for weight-2 errors and coded data)	
...	...	...

Table 1: PowerPC Address Space

parameters that the user may alter include:

- How long the test is run. The user can either choose to end the test manually or to end the test after a specified number of bit errors have occurred.
- Which code is enabled. The user can choose to enable the Hamming, BCH, SEC-DED, or Fire coder/decoder, or coding can be disabled entirely.
- The length of the codeword. Any length that is a multiple of forty may be selected if an uncoded sequence is run. A coded sequence, however, only works if a block size of 40, 80, 480, or 1000 is selected.
- The RocketIO's transmitter and receiver equalizer settings. The user may change the default register settings for the equalizers. Xilinx also recommends certain settings for other RocketIO registers depending on the desired data rate. The user may also choose to use any one of these recommended settings.

At the end of a test, the PowerPC reads the test results from the registers of the RocketIO peripheral, including the number of 40-bit words that were received, the number of total bit errors that occurred, and all of the RAM addresses. From the data, the PowerPC can calculate the bit error rate. Then, it formats and displays all of the results to the user. The RAM data is formatted into both MATLAB-compatible vectors and printed for easy viewing.

5.5 Codes Implemented

The codes chosen for study in the project are Hamming, BCH, Fire, and SEC-DED codes. These codes are chosen, because they are effective at combating different types of errors and are used in different types of systems. The structure and properties of each code are briefly discussed in the background section. A more thorough explanation can be found in [7].

5.5.1 Simplifications

Due to limitations imposed by the hardware, a few simplifications have been made to the choice of codes and their implementa

Another simplification made due to hardware constraints is the implementation of decoders. The correction circuit for simpler codes are implemented in full. However, the correction circuits of more powerful codes are extremely complex. Difficulties arise from the requirement that 40 bits be generated or received at each clock cycle. There are several techniques to handle parallel data for some known codes. However, these simpler techniques often require that the code is cyclic, which does not apply to the shortened codes implemented in the design. It is impractical to implement some of these circuits, because it is not necessary to the evaluation of the code. Post-processing the error statistics of a coded sequence is sufficient for determining the approximate BER.

5.5.2 Hamming Codes

Hamming codes are a class of simple codes used for single-bit error correction. Hamming codes are evaluated in the project, because they have high code rates, and single-bit errors dominate the total errors through the link even for very long codeword lengths. Thus, even single-bit error correction may be able to improve the bit error rate when compared to uncoded data with equal information throughput.

The same codes may also be used to detect errors in an ARQ protocol. Hamming codes have a Hamming distance of three, so they are guaranteed to detect all single- and double-bit errors within a codeword. They are also able to detect many other error patterns, especially since the code is shortened.

To conform to the natural 40-bit block size of the hardware, all the Hamming codes must be shortened, which no longer makes them perfect. The three Hamming codes are arbitrarily chosen to be the three primitive polynomials with the fewest terms possible for their degree. If implemented serially, these polynomials would result in circuits with the least number of taps. Their generator polynomials are:

- $\mathbf{g}(X) = 1 + X + X^6$ for the (40, 34) Hamming code shortened from (63,57).
- $\mathbf{g}(X) = 1 + X^4 + X^8$ for the (80, 73) Hamming code shortened from (127,120).

- $g(X) = 1 + X^3 + X^{10}$ for the (1000, 990) Hamming code shortened from (1023,1013).

5.5.3 BCH Codes

BCH codes are similar to Hamming codes, but generalized for multiple-bit error correction. BCH codes may provide better performance over Hamming codes, because the bursty nature of the link causes multiple errors in a codeword to occur more frequently than in a binary symmetric channel. However, they also require much more redundancy to achieve the improved results.

Three 2-error-correcting BCH codes are implemented with codeword lengths of 40, 80, and 1000 to directly compare results with the Hamming code and the other codes that are studied. Furthermore, since up to five errors per codeword occur rather frequently, a 5-error-correcting (480,435) BCH code is also implemented. The block size of 480 was chosen for its high code rate of 0.906.

The Hamming distance of the 2-error correcting codes is at least five, so these codes may also be used to detect all error vectors of weight less than or equal to four. The (480,435) shortened BCH code is guaranteed to detect all error vectors with weight at most 10. Thus, BCH codes are extremely powerful at detecting errors. Since nearly all words have fewer than nine errors, these BCH codes should be very effective at error detection.

The generator polynomials for the four shortened BCH codes are:

- $g(X) = 1 + X^3 + X^4 + X^5 + X^8 + X^{10} + X^{12}$ for the (40,27) 2-error-correcting BCH code shortened from (63,51).

- $g(X) = 1 + X + X^2 + X^4 + X^5 + X^6 + X^8 + X^9 + X^{14}$ for the (80,66) 2-error-correcting BCH code shortened from (63,51).

- $g(X) = 1 + X + X^2 + X^4 + X^5 + X^6 + X^{11} + X^{12} + X^{20}$ for the (1000,980) 2-error-correcting BCH code shortened from (1023,1003).

- $g(X) = 1 + X + X^4 + X^5 + X^6 + X^7 + X^8 + X^9 + X^{13} + X^{14} + X^{15} + X^{19} + X^{22} + X^{23} + X^{25} + X^{26} + X^{29} + X^{31} + X^{33} + X^{34} + X^{35} + X^{37} + X^{39} + X^{42} + X^{45}$ for the (480,435) 5-error-correcting BCH code shortened from (511,466).

5.5.4 Fire Codes

Fire codes are the first systematically generated single-burst-error-correcting codes. Fire codes are studied in the project, because some of the noise sources in links affect the data stream for multiple bits. Thus, a portion of the errors are correlated and cause bursts.

Fire codes are also studied because they have been shown to be effective for burst-error-correction, burst-error-detection, and simultaneous burst-error detection and correction. Thus, the codes implemented can be analyzed for use in all three error-correction protocols. The codes implemented in the design have $c = 7$, so they are capable of correcting all bursts of four, detecting all bursts of eight, or simultaneously correcting single-bit errors and detecting bursts of seven, depending on how the decoder is implemented. These codes are capable of simultaneously correcting and detecting other combinations of burst errors, but these further combinations are not studied.

One more Fire code is also implemented, a (480,425) shortened code, in order to analyze the performance of a more powerful, high-rate Fire code. The (480,425) shortened Fire code has $c = 35$, so it is able to correct all bursts of 18, detect all bursts of 36, or simultaneously correct and detect many combinations of error lengths.

The generator polynomials implemented in the project are:

- $g(X) = (1+X^7)(1+X+X^6) = 1+X+X^6+X^7+X^8+X^{13}$ for the (40,27) 4-burst-error-correcting Fire code shortened from (63,50).

- $g(X) = (1+X^7)(1+X+X^4) = 1+X+X^4+X^7+X^8+X^{11}$ for the (80,69) 4-burst-error-correcting Fire code shortened from (105,94).

- $g(X) = (1+X^7)(1+X^2+X^3+X^4+X^8) = 1+X^2+X^3+X^4+X^7+X^8+X^9+X^{10}+X^{11}+X^{15}$ for the (1000,985) Fire code shortened from (1785,1770).

- $g(X) = (1+X^{35})(1+X^{15}+X^{20}) = 1+X^{15}+X^{20}+X^{35}+X^{50}+X^{55}$ for the (480,425) 18-burst-error-correcting Fire code shortened from (525,470).

5.5.5 Single Error Correction, Double Error Detection (SEC-DED)

SEC-DED codes have several characteristics that make them potentially suitable for links. They require only one extra parity bit than the Hamming code of the same codeword length, so they have high code rates. SEC-DED codes are able to handle double errors, which Hamming codes are forced to miscorrect. Thus, SEC-DED codes potentially have significant performance gains for low additional overhead. Another beneficial property of SEC-DED codes is the detection circuit that signals a retransmission in an ARQ system. The circuit simply checks whether the syndrome of the received vector is even, rather than having to determine whether a non-zero syndrome is a correctable error pattern before requesting a retransmission. This form of detection circuit is unable to detect odd-weight error vectors that are not correctable, but the simple implementation to an otherwise complex problem is very desirable.

Unlike all of the other codes tested in the project, SEC-DED codes are not cyclic. They are defined only by their parity-check matrices. To conform to the block sizes of the previously discussed codes, the SEC-DED codes implemented have parity-check matrices that generate (40,33), (80,72), and (1000,989) codes. The codes, along with the weight of each column and the number of columns of each weight, are shown below. For example, the first listed code, the (40,33) SEC-DED code, has seven columns of weight one and 33 columns of weight three.

- (40,33) code generated with parity-check columns of weight: $\binom{7}{1} + 33/\binom{7}{3}$
- (80,72) code generated with parity-check columns of weight: $\binom{8}{1} + \binom{8}{3} + 16/\binom{8}{5}$
- (1000,989) code generated with parity-check columns of weight: $\binom{12}{1} + \binom{12}{3} + \binom{12}{5} + \binom{12}{7} + 184/\binom{12}{9}$

SEC-DED codes, as described by Hsiao in [11], are designed to have the fewest number of 1's possible in the parity-check matrix, which results in the shortest logic delay in generating parity bits and syndrome digits. Furthermore, the columns of greatest weight are chosen in such a way to have an equal number of 1's per row. Based on the design of the hardware, which breaks the codeword into groups of 40 bits, the project adds the constraint that the 1's are distributed evenly between each block of 40 bits in a row.

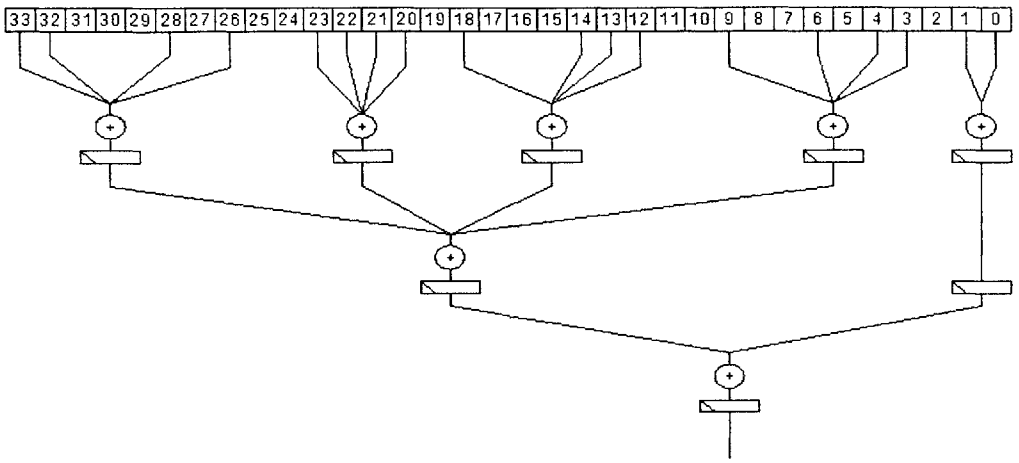

Figure 16: Encoder Circuit for the Last Parity Bit in the (40,34) Hamming Code

5.6 Encoder Design

The encoder consists of several modules, one for each code implemented. Since every code can be defined by a generator or parity-check matrix, the encoder simply realizes the generator matrix in hardware. The encoder must implement the equation, $\mathbf{v} = \mathbf{u} \cdot \mathbf{G}$, where $\mathbf{u}$ is the information vector and $\mathbf{G}$ is the generator matrix. Since all the codes are in systematic form, the information digits remain unchanged and in order, and only the $n-k$ redundant bits must be generated. Each parity bit is generated by xor-ing a subset of the information bits. The specific bits used in the equation are determined based on a column of generator matrix. For example, the circuit that generates the final parity bit in the (40,34) Hamming code looks like the circuit in Figure 16. The corresponding generator matrix column, not including the identity matrix, is [1000011000101001111010001110010010].

The parity bits are formed in three cycles to maximize FPGA resources. Each FPGA slice has two 4-input look-up tables (LUTs) and two registers, so the circuit utilizes each cell as much as possible. For block sizes larger than 40, a sequence of n bits must be made available from the PRBS generator before the parity bits may be formed. Only k of the n bits are actually used as information bits, and the remaining bits are replaced with the generated parity bits. The generic encoder structure for codes with $N > 40$ is shown in Figure 17.

Once the parity bits are formed, they must be inserted into the proper location in the codeword.

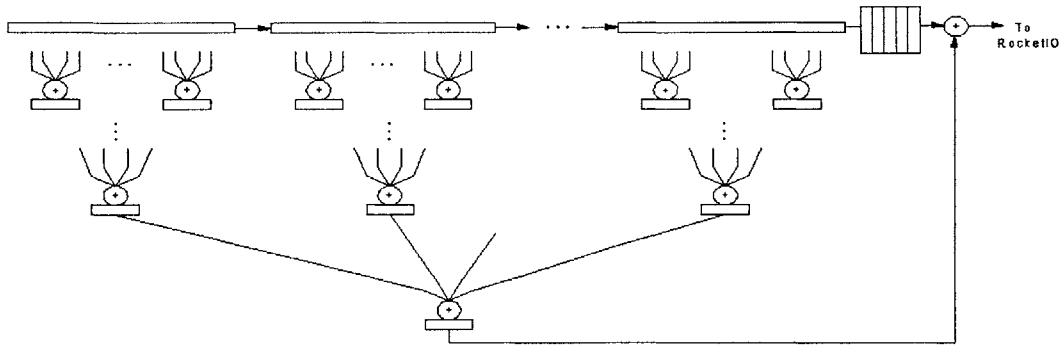

Figure 17: A Generic Encoder for a Code with More Than 40 Bits

For the Hamming coder partially diagrammed in Figure 16, the redundant bits are not available until three cycles after all of the information bits have entered the registers. Therefore, instead of immediately sending the information bits to the RocketIO on the next cycle, the entire word must be delayed three cycles. This is true for the generalized case of an arbitrarily long codeword. Generally, there must be a delay block after the final register in Figure 17 that is the same number of cycles as the number of cycles it takes to create the parity bits. The generated parity bits will then be valid at the same cycle as the last 40-bit block is ready to enter the RocketIO transmit register. The parity bits can then be inserted into the last digits of the block. The block number tells the encoder which cycle to insert the parity bits.

For the (480,425) Fire code and the (480,435) BCH code, since the number of parity bits exceed 40, the parity bits must be inserted into the final two 40-bit blocks. Only a minor adjustment is required in this case. The circuit must be synchronized such that the parity bits are valid the same cycle as the second to last 40-bit information block of the n-bit sequence. The first $n-k-40$ parity bits replace the last $n-k-40$ bits in the 40-bit word. The last 40 redundant bits are delayed a cycle and sent as the last 40 bits in the n-bit codeword.

5.7 Decoder Design

To fully decode each code, every code must have a syndrome circuit and a correction circuit. The syndrome circuit signals that an error has occurred if the result is non-zero. The correction circuit then uses the value of the syndrome to flip the appropriate bits.

The syndrome circuit is virtually identical to the encoder circuit. The algorithm and the implementation are exactly the same. The minor difference is the syndrome circuit forms the syndrome from n bits while the encoder forms the redundant bits from only the k information bits. Thus, the syndrome has one extra bit to include in the xor equation for each syndrome bit.

As stated previously, most of the correction circuits for the various codes are left unimplemented. The exceptions are the $n = 40$ and $n = 80$ Hamming and SEC-DED codes. Both of these codes implement single-bit error correction, which can be implemented without a large amount of logic, since there are only n error vectors that the codes must be able to correct. The syndrome is calculated based on the equation, $\mathbf{s} = \mathbf{r} \cdot \mathbf{H^T}$, where $\mathbf{r}$ is the received vector and $\mathbf{H}$ is the parity-check matrix. Therefore, when a single-bit error occurs at position i in the received vector, the syndrome is equal to the ith column of $\mathbf{H}$. The correction circuit for a Hamming or SEC-DED code determines whether the syndrome is equal to any of the columns of the parity-check matrix. If it is equal to the ith column than the ith bit in the received vector is flipped. For example, the last column in the parity-check matrix of the (40,34) shortened Hamming code is [011011]. Thus, the correction circuit for the least significant information bit can be implemented according to the equation:

$\mathbf{c}(0) = \mathbf{r}(0)$ xor (not($\mathbf{s}(5)$) and $\mathbf{s}(4)$ and $\mathbf{s}(3)$ and not($\mathbf{s}(2)$) and $\mathbf{s}(1)$ and $\mathbf{s}(0)$).

There are 40 such circuits to realize the full correction circuits for the (40,34) Hamming and (40,33) SEC-DED codes, and 80 such circuits in the (80,73) Hamming and (80,72) SEC-DED encoders. Extra registers are required to create a more FPGA-friendly correction circuit than the simple equation above.

The complete decoder circuit for one code is pictured in Figure 18. Like the encoder circuit, the output of the syndrome circuit is only valid every $n/40$ cycles, though it changes every cycle. The correction circuit, however, requires that the syndrome is constant and valid while it performs the correction. The register between the syndrome circuit and the correction circuit saves the valid syndrome for the full $n/40$ cycles until the next one is calculated. Again, like the encoder circuit, the data from the link must be delayed for extra cycles until the syndrome is calculated and is passed through the register to the correction circuit. The shift register shown in Figure 18 delays the data such that the data from the link and the valid syndrome are synchronized to the same cycle. The block number must also be synchronized

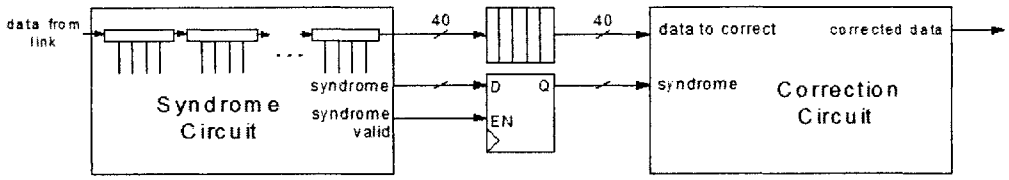

Figure 18: Decoder Circuit

so that the correction circuit can differentiate between codewords.

Notice that Figure 18 only shows the decoder for a single code. To create the entire decoder, this circuit, with the necessary changes to the syndrome and correction circuits, can be copied 10 times for a total of 11 separate decoders for the 11 different codes. However, this strategy results in excess hardware. Some hardware components may be shared between codes, such as the shift registers, since only one code is enabled at a time. Thus, the decoder has several syndrome circuits feed into the same shift register, which feeds into multiple correction circuits. The correct circuit to use is chosen using a multiplexer. The shift registers, therefore, have variable delay which generally depends on the codeword length.

6 Results

6.1 Tuning the Equalizer and Choosing Data Rates

The links are tested at a couple of appropriate rates to get an idea of the error distribution at different points in the frequency spectrum. These rates are selected based on a desired bit error rate, which is ideally around 10^{-6} or 10^{-7}. If the BER is much lower than that, then it is impractical to obtain a reasonable amount of bit errors in a test to be confident of the true distribution, especially after coding is applied. Furthermore, the goal of the project is to improve the performance of links when the BER is fairly high. On the other hand, if the bit error rate is much higher than 10^{-6}, most likely the errors are being caused by a complete inability to properly equalize the signals, which is not the case of interest in the project.

To obtain the BER at a given data rate, the equalizers must be set appropriately. An exhaustive test of all configurations is used to find the best combination of the transmit and receive equalizer settings

Data Rate (Gb/s)	Bit Error Rate
5.00	3×10^{-12}
5.25	1×10^{-10}
5.50	8×10^{-10}
5.75	3×10^{-9}
6.00	2×10^{-7}
6.25	2×10^{-6}
6.50	1×10^{-4}
6.75	1×10^{-3}

Table 2: Bit Error Rate for Various Data Rates on a Rogers Link

described in the RocketIO X section of the hardware description. Every combination is run until 100 bit errors are observed. At the end of the search, the combination resulting in the longest test is assumed to be the best setting.

For the 20 inch Rogers link, the test produced the results shown in Table 2. As can be seen from the table, the bit error rate is extremely low at data rates below 6 Gb/s and then falls drastically at rates higher than 6.25 Gb/s. Thus, for the remainder of the tests, only data rates of 6 and 6.25 Gb/s are used.

6.2 Results Compared to a Binary Symmetric Channel

Figure 19 shows the probability of having x bit errors per word for reasonably small x based on experimental data for a wide range of data rates. This is compared with the projected probabilities for a BSC. For high data rates where the bit error rates are very low, the channel is similar to the binary symmetric channel model. Although it is not possible to determine why the link produces these results with 100% confidence, it is likely that reflections are the dominant noise sources, and the equalizers are not able to combat them. Reflections are the likely candidate, because the backplane is not counterbore, and reflections are generally uncorrelated.

For data rates of interest, such as at 6 Gb/s as stated in the previous section, the channel curve deviates drastically from the BSC curve. To obtain bit error rates of lower than 10^{-15} using random-error correction, a BSC would need only a 2-error-correcting code. To obtain an adequately low BER for the link may not even be possible with simple random-error correction. The channel curve indicates that at least some of the dominant noise sources are correlated. Thus, it is very unlikely that the existing

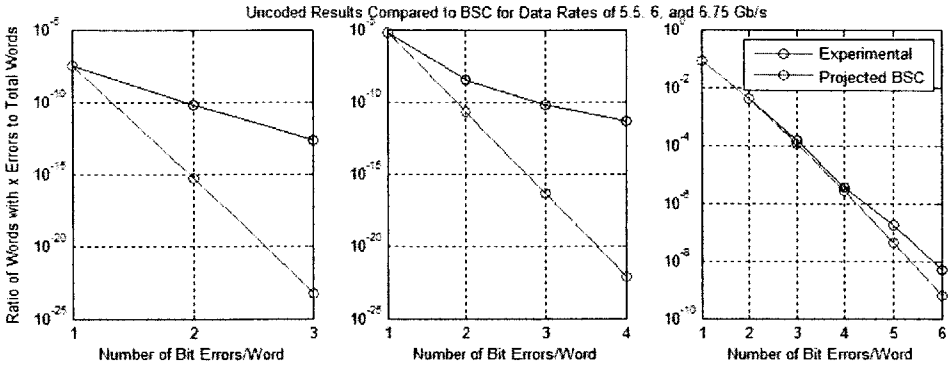

Figure 19: The three graphs show the behavior of the link at three different data rates: 5.5 Gb/s (left), 6 Gb/s (middle), and 6.75 Gb/s (right). As the data rate increases, the link becomes more and more like a BSC.

error-correcting codes chosen for study will be capable of bringing the BER down to the desired 10^{-15}. However, these codes may give an indication of what kinds of codes to try in the future to achieve the BER goal. Since the link is not a BSC, any information must be determined experimentally.

6.3 Uncoded Results

6.3.1 Number of Errors Per Word for Each Data Rate and Block Size

The left half of Figure 20 shows the total number of x-bit errors that occurred for tests run at different block sizes at data rate of 6 Gb/s. The equivalent graphs for a data rate of 6.25 Gb/s is shown on the right half of the figure. As can be seen by these figures, the number of n-bit errors per block stays relatively constant for block sizes between approximately 100 and 1000. Generally speaking, code rates increase dramatically as the codeword length increases. For example, the shortest full-length Hamming code has a rate of $4/7 \approx 0.57$ while a (1024,1014) Hamming code has a code rate of approximately 0.99. Therefore, it may be worth increasing the codeword length in order to use a more powerful code. Increasing the block size, however, increases the required hardware of the transceiver. This tradeoff must be thoroughly examined.

Based on these results, three codeword lengths are chosen for testing: 40, 80, and 1000. Currently, block sizes of 40 are very typical, and $n = 80$ is on the very high end of what has been attempted in links. Finally, assuming the number of bit errors per word is constant for coded data streams as well as

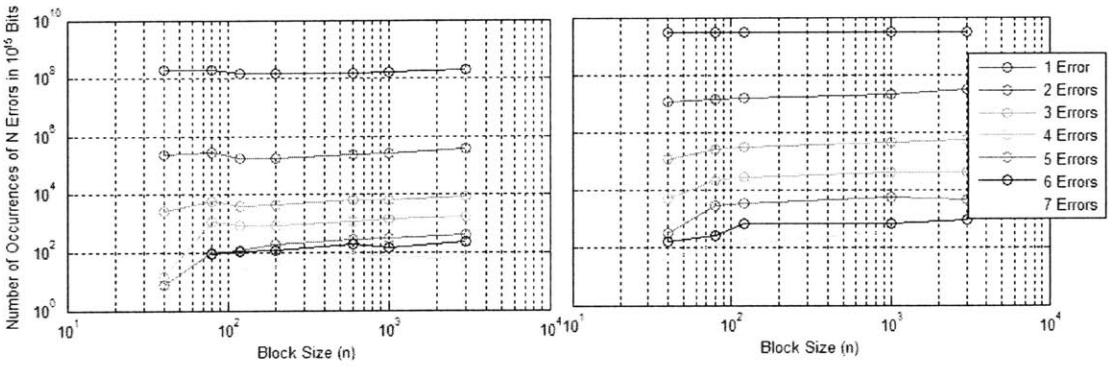

Figure 20: Distribution of Errors Per Word at 6 Gb/s and 6.25 Gb/s: The graphs show the number of x bit errors per word over multiple block sizes that occurred for x between one and seven. The graph on the left is for 6 Gb/s, and the right is for 6.25 Gb/s. Each column of circles in the graph indicates a separate test run through the link. Although there are a few instances of words with more than seven errors, not enough occurrences were observed to accurately predict a trend. The y axis is normalized to 10^{15} bits. The results are actually based on 10^{14} to 10^{15} bits for 6 Gb/s and 10^{12} to 10^{13} bits for 6.25 Gb/s.

uncoded streams, $n = 1000$ will give the expected bit error rate for block sizes between approximately 100 and 1000. Also, it is useful to compare the BER of very large block sizes with that of small lengths to examine the potential of using the codeword length as an important variable in transceiver design.

6.3.2 Error Lengths for Each Data Rate and Block Size

Since the channel is not a BSC, the distribution of error lengths reveals some information on the correlation between the errors. The distribution of error lengths in a 2×10^{13}-bit test at 6.25 Gb/s is shown in Figure 21, with results normalized for an even 10^{14} bits. For 6 Gb/s, the results are very similar.

Because the set of graphs is on a logarithmic scale, it is not obvious that a significant proportion of the errors have bursts of fewer than 10 bits. From a closer examination, it is clear that the bars associated with lengths two through six on the top graph are significantly higher than any other bar. At 6 Gb/s, 99.94% of all bit errors occur in codewords with error lengths of less than 10. Most of these, however, are single-bit errors, which technically have error lengths of one. For words with multiple errors, 75% of the bit errors are still within a length of nine. At 6.25 Gb/s, these numbers are 99.90% and 87.21%, respectively.

For block sizes larger than 40, the picture looks rather similar to Figure 21. As can be expected, when

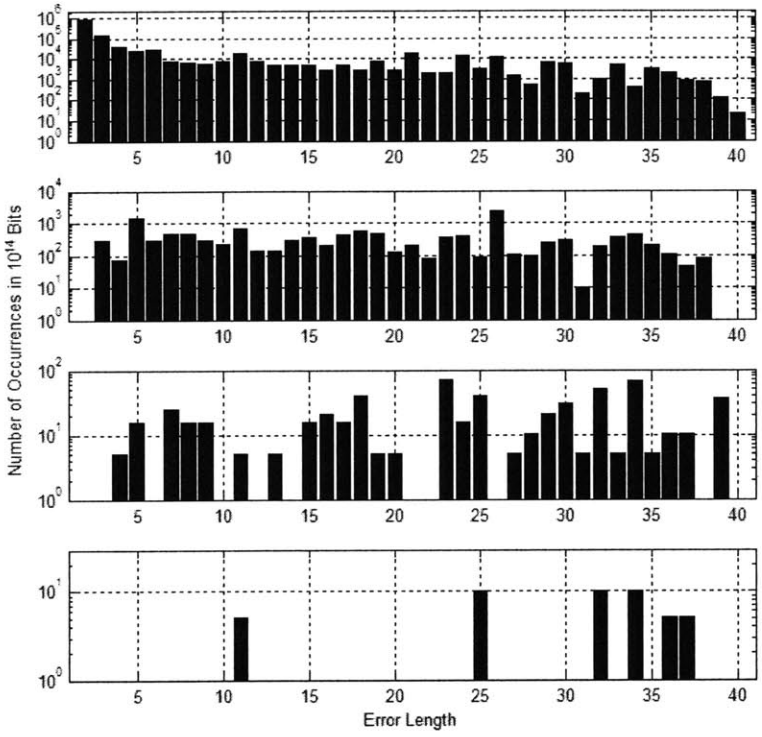

Figure 21: Distribution of Error Lengths at 6.25 Gb/s on a Rogers Link. The error lengths of all 40-bit words with multiple errors is illustrated for a single test run, normalized to 10^{14} bits. The graphs indicate the error length distribution of error patterns of weight two (top), three (upper middle), four (lower middle), and five (bottom).

the block size is increased, errors that would otherwise have shown up in adjacent words end up appearing in the same word. Thus, as the block size increases, the percentage of blocks in error with short bursts decreases slowly. The effectiveness of a burst error correcting code will likely diminish. However, the number of parity bits required of a burst error correcting code is generally dependent on the burst error correcting capability of the code, and not necessarily on the size of the block. The number of parity bits needed for random error correction, like for Hamming or BCH codes, increases monotonically with block size. Thus, since the code rate may be better with burst error correction than random error correction at higher block sizes, it has potential to be effective against link noise.

One other notable conclusion that can be drawn from the graphs that make burst error correction less appealing are that words with a larger number of errors generally have large lengths. From the graph, only words with two bit errors have consistently short lengths. For four bit errors, most errors are at least of length 23, and for five bit errors, the shortest length is 11. Thus, burst error corrections will likely be ineffective for words received with more than a couple of errors. Since words with multiple errors occur relatively frequently, a code that utilizes only burst error correction will not likely be more successful at obtaining transmission reliability than a random error correcting code.

6.4 Coded Results

The following sections detail the results observed using the codes mentioned previously. Each code's performance is analyzed with respect to the equivalent uncoded data and with respect to hardware requirements and complexity. Finally, the results of all of the codes are summarized and directly compared. From the comparison, a conclusion may be reached on how and what type of coding has the most potential for use with links.

Before all of this can be done, first the simplifications and approximations used when analyzing the data are described.

6.4.1 Simplifications and Approximations Used for Generating Results

There are two simplifications used when generating the following results for the codes. Neither should affect the results significantly. However, if these or similar approximations are used in future projects, their effect on the results should be double checked to guarantee that the conclusions made are valid.

The first approximation is for generating results when no correction circuits are used, which is needed for larger block sizes and for more complex codes. The results of correction can be approximated by simply not counting errors that occur within blocks that should be corrected. For example, if a single-bit error occurs in a codeword when a Hamming code is used, that error is assumed to be corrected accurately. The approximation deviates from the actual value whenever a mis-correction takes place. For example, if two errors occur when a Hamming code is used, the actual correction circuit may miscorrect a bit, therefore generating three bit errors in that codeword. The approximation misses that extra bit error. Thus, the actual BER is generally higher than the estimated BER. However, the estimation only misses at most one error whenever an uncorrectable error pattern occurs, meaning that the actual BER is at most double the estimated BER. Since it is the order of magnitude of the BER that is of interest in the project, the approximation gives satisfactory results.

The second approximation is for generating detection results when no ARQ protocol is set up. For the purposes of the project, it is assumed that if an error is properly detected, it will eventually be retransmitted and accepted without errors. Furthermore, the approximation does not include the decrease in the throughput that results from one or more retransmissions. The latter approximation is used since the round trip delay of links is relatively short. For 10 Gb/s, the round trip delay is less than 100 bits. As an example, assuming for the moment that the channel is binary symmetric, using a go-back-N retransmission scheme at 6.25 Gb/s would result in a throughput of 0.996 times the throughput for a FEC scheme. Go-back-N is an ARQ protocol that continuously transmits bits to the receiver, and it is the continuous transmission scheme that has the lowest throughput and lowest complexity. Since it has been shown that the link is somewhat bursty, the throughput will be somewhat altered. In the future, it will be necessary to determine what the actual throughput is. However, the throughput of the BSC case is high enough for the assumption to be warranted, especially since any decrease in the throughput can

be compensated for by choosing a different ARQ protocol.

The first part of the assumption, which is that detected errors will always be retransmitted without errors, results in a BER estimation that will likely be lower than the actual BER. The main problem with this assumption is that since the link is not a BSC, a codeword that is received with detectable errors is far more likely than other codewords to incur errors a second time. To illustrate this point, the following 80-bit pattern within the PRBS is very likely to cause errors:

11011000000000000000111110100001100000000000000000011100110110000000000000000000

Notice that the DC balance and number of transitions are far from ideal. The 80-bit pattern is isolated by repeatedly running the pattern through the link. The bit error rate is approximately 10^{-2}. This appears to show that it is likely for a retransmitted word to eventually be accepted with errors. However, whenever errors occur, they almost always form the same, or same couple of, error patterns. Remember that for a codeword to be accepted by the decoder, an error pattern that is a codeword must occur. Therefore, the main issue with worst case patterns is that they may have to be retransmitted multiple times before no errors occur. This is not a major problem as long as none of the worst case patterns fully close the eye. If the eye is completely closed, the sequence will never be accepted by decoder, and the system cannot continue. Otherwise, having to retransmit sequences multiple times does not affect the BER of the decoded data. It only affects the throughput of the system.

6.4.2 Results for Hamming Codes

Hamming codes are implemented in the project, because they could potentially attain equal or better reliability than uncoded data rate at the same information rate. If a code this simple is capable of achieving any benefits over the uncoded case, then it is likely that there exists a much more powerful code that can greatly enhance the performance of the link. From the data, it can be seen that Hamming codes are indeed able to perform better than the uncoded case for large enough block sizes.

Effect of Redundancy on Bit Stream's Error Distribution As previously explained, (40,34), (80,73), and (1000,990) Hamming codes were implemented and tested. Based on the results of running tests at 6 Gb/s and 6.25 Gb/s, the graphs in Figures 22 and 23 were generated.

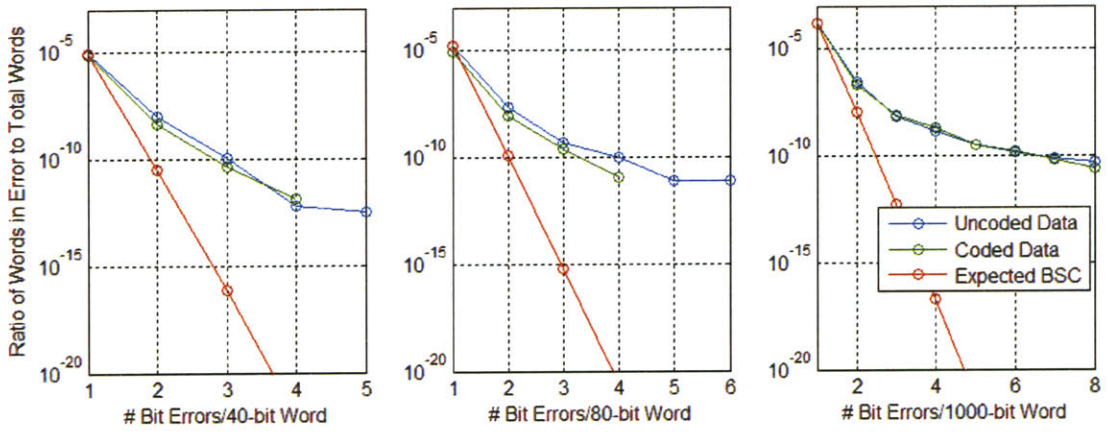

Figure 22: Data Dependency of Coded Hamming Data Stream at 6 Gb/s

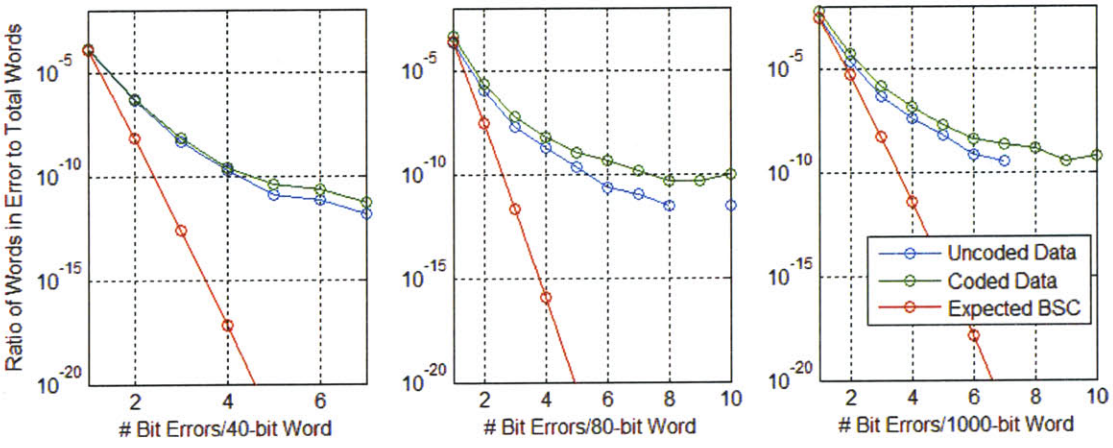

Figure 23: Data Dependency of Coded Hamming Data Stream at 6.25 Gb/s

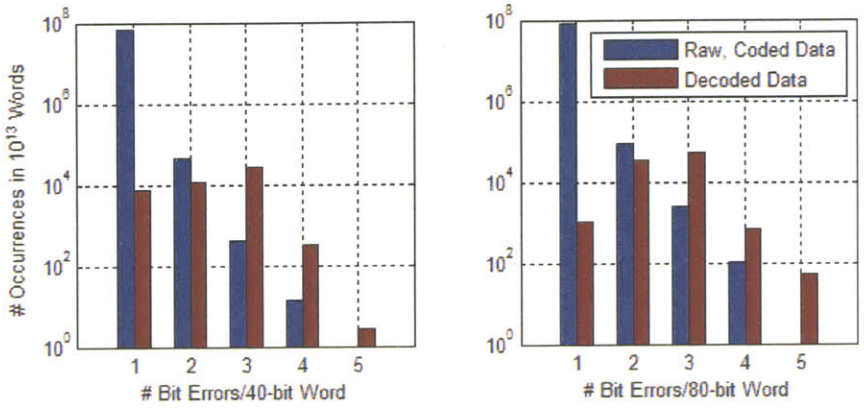

Figure 24: Results of Hamming Codes Before and After Correction for N=40 (left) and N=80 (right) at 6 Gb/s. The raw, coded data is based on the entire codeword, or n bits. The decoded data is based on the information digits only, or $k = n - k$ bits per word.

For the most part, the coded data stream produces an error distribution that matches the uncoded data. The two curves match extremely well when there are only one or two errors in the word. For Hamming codes, 2-bit errors dominate the BER. Thus, the performance of Hamming codes may be accurately predicted by the error distribution of the uncoded data stream.

Results of Error Correction Using Hamming Codes The error distribution after correction for the (40,34) and (80,73) Hamming codes is shown in Figure 24 for 6 Gb/s. The results for the Hamming codes are based on tests of 3.6×10^{12} and 3.75×10^{11} codewords, respectively, and both normalized to 10^{13} words. The figures show the effect of mis-correction on the data stream. Due to mis-correction, many of the codewords with x errors end up with $x + 1$ errors. In the figures, the number of errors before correction takes place is calculated based on the entire n-bit code, including the m parity bits. For the corrected data, however, only the information bits are considered. Thus, if errors occur in the parity bits because of the link or from mis-correction, they are not tallied. For example, a 3-bit error with two errors in the information digits will result in a 2-bit error if a parity bit is mis-corrected (or if the decoder does nothing, which can occur for shortened Hamming codes only) and a 3-bit error if an information bit is incorrectly flipped. Notice that, as expected, the total number of pre-corrected codewords with more than one error is approximately the same as the total number of decoded words with errors.

As can be seen on the left of Figure 25 for 6 Gb/s, represented by circles, the bit error rate does

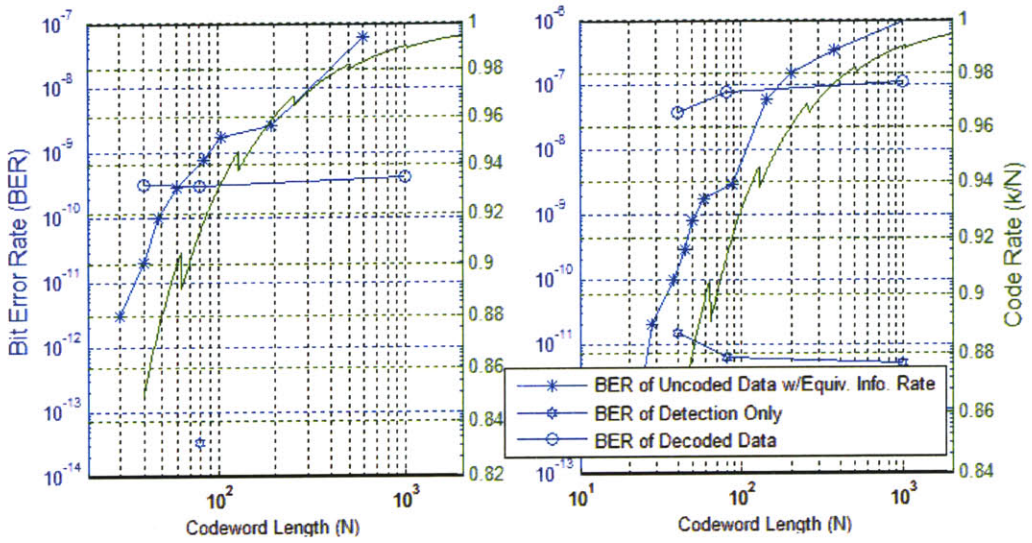

Figure 25: Effectiveness of Hamming Codes with Block Size of 40, 80, and 1000. The left graph shows results for 6 Gb/s, while the right graph is for 6.25 Gb/s. The raw bit error rates are approximately 2×10^{-7} and 4×10^{-6}, respectively. The BER for block sizes of 40 and 80 are based on actual correction circuits, while the BER for $N = 1000$ is an approximation of the actual BER.

not significantly increase when the block size increases. Recall that the bit error rate of the data before correction is 2×10^{-7}. After correction, the average BER is approximately 4×10^{-10}, a raw improvement of almost three orders of magnitude.

In order to accurately judge the effectiveness of the code, the BER of the decoded data must be compared to uncoded data transmitted at the same information rate. The corresponding information rate depends directly on the code rate, so the (1000,990) Hamming code has a significant advantage over the other two codes. The (40,34) Hamming code must be compared to uncoded data running at 5.1 Gb/s, while the (1000,990) Hamming code can be compared to data transmitted at 5.94 Gb/s.

Both graphs in Figure 25 also show the code rate for Hamming codes as a function of the block size. Notice that the line jagged. For a fixed number of parity bits, m, the more shortened a code is, the lower its code rate. Thus, the rate curve has a local maximum whenever a full block size of $2^m - 1$ is used, and then immediately falls to a local minimum since the code is the most shortened possible for $m + 1$ parity bits. Based on the code rate, it is easy to see why the BER of the uncoded data, marked with asterisks, rises steadily with block size. In fact, below block sizes of approximately 60, using a Hamming

code actually decreases the performance of the link. If the block size is increased all the way up to 1000 or higher, it is likely that the BER improvement is quite close to the raw improvement of three orders of magnitude.

The equivalent graph for 6.25 Gb/s is shown on the right of Figure 25. At 6.25 Gb/s, since more multiple-bit errors occur, the maximum improvement of coding is only 1.5 orders of magnitude, from approximately 4×10^{-6} to 8×10^{-8}. Coding does not provide any gain over sending uncoded data unless block sizes of approximately 200 or greater are used.

Results of Error Detection Using Hamming Codes Figures 25 has further information that shows the approximate BER if the Hamming code is used as a CRC (detection and retransmission is used). Using detection, the graph shows that coding is beneficial in terms of performance at all block sizes. At large block sizes, using detection, even with such a simple code, a BER improvement of more than five orders of magnitude is observed at 6.25 Gb/s. The raw BER of approximately 10^{-6} drops down to 10^{-11}. At 6 Gb/s and $N = 80$, the test was run long enough to observe a few errors. In this case, the detection circuit is an improvement over the correction circuit by four orders of magnitude, from a little over 10^{-10} to a little over 10^{-14}. These results are particularly interesting, because it shows that with detection, a code may not need to be very powerful in order to detect enough errors for 10^{-15} reliability.

6.4.3 Results for SEC-DED Codes

SEC-DED codes are an improvement on Hamming codes to enable detection of all double-errors. For the detection case, we make the assumption, as explained in the previous section describing simplifications, that any errors that are detected will eventually be received correctly.

Effect of Redundancy on Bit Stream's Error Distribution The results generated by running SEC-DED codes through the link are remarkably similar to the results for Hamming codes, as shown in Figures 22 and 23. With the way the hardware is implemented, only the m bits of the SEC-DED code differ from that of the Hamming codes. The remaining information bits are the same. For block sizes of 1000, only eleven bits in the entire block may have different values. Thus, any differences may not be

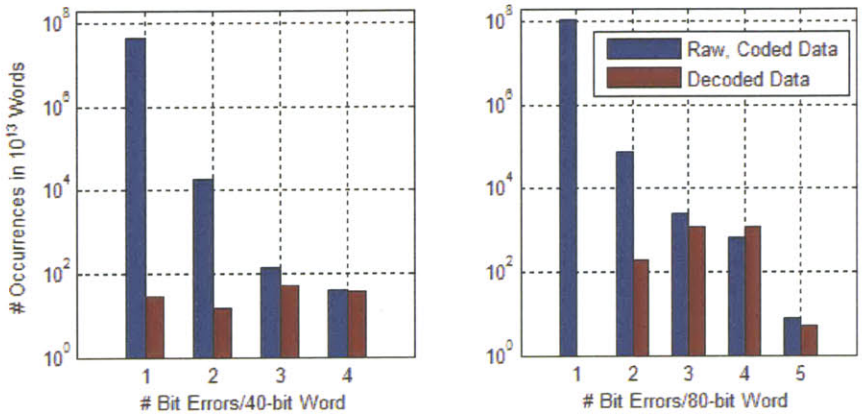

Figure 26: Results of SEC-DED Codes Before and After Correction for N=40 (left) and N=80 (right) at 6 Gb/s. The raw, coded data is based on the entire codeword, or n bits. The decoded data is based on the information digits only, or $k = n - k$ bits per word.

significant enough to affect the ISI.

However, the two curves representing the uncoded and coded data deviate as the number of errors per word increases. Much of this is possibly due to the fact that fewer occurrences were observed, so the results may not be an accurate reflection of the true proportions. For a code that handles two errors, such as the SEC-DED code, the deviation is not particularly extreme, so one may still obtain a reasonably accurate prediction of the effectiveness of SEC-DED codes based on the uncoded data alone.

Results Obtained by the SEC-DED Codes As with Hamming codes, the results of decoder at 6 Gb/s are shown in Figure 26. The tests consist of 2.7×10^{12} codewords and 8.1×10^{12} codewords, respectively, but are normalized to 10^{13} words for graphing purposes. The figure shows the error distribution of the errors before and after entering the decoder. The decoder is implemented such that it can detect all even-weight error patterns and correct all single-bit errors. Thus, in the figure, the total sum of the odd-weight columns greater than one of the pre-decoded data is approximately the same as the total sum of the columns in the decoded data. From the figure, it is clear that there are still a significant amount of triple-errors that the code is forced to miscorrect. These triple-errors easily prevent the code from being as effective as desired.

The results for the comparison between SEC-DED codes and uncoded data at 6 Gb/s and 6.25 Gb/s is

shown in Figure 27. For both data rates, the BER is fairly constant for block sizes greater than 80. Notice also that the uncoded data curve, marked with asterisks, is shifted slightly to the right compared to the corresponding Hamming graph. The code rate for SEC-DED codes is slightly lower, so the uncoded data must be run slightly faster in order to maintain the same information rate. Even though the SEC-DED code has to make up more ground, because it corrects double-errors, which happen very frequently (see Figures 24 and 26), its performance is better. At 6 Gb/s, SEC-DED codes outperform the uncoded data for every block size tested. The BER of the decoded data is about 10^{-11}. The raw BER of uncoded data is about 2×10^{-7}, so the SEC-DED code improves the reliability by more than four orders of magnitude at large block sizes. Even at a block size of 80, the code is an improvement over uncoded data by more than an order of magnitude, from 3×10^{-10} to 10^{-11}.

At 6.25 Gb/s, the SEC-DED code outperforms uncoded data above block sizes of approximately 100. The BER of the decoded data is stable at about 4×10^{-9} for block sizes around 80, so the effectiveness of the code monotonically increases with the block size starting from that point. For a block size of 1000, the (1000,989) SEC-DED code is able to improve the BER from 10^{-6} to 4×10^{-9}, an improvement of approximately 2.5 orders of magnitude.

6.4.4 Results for BCH Codes

BCH codes with t=2, like SEC-DED codes, are able to handle all single-bit and double-bit errors within a codeword. The difference between the two codes is that BCH codes correct double errors instead of just detecting them. The increased ability to correct is obtained by using more bits for redundancy.

Effect of Redundancy on Bit Stream's Error Distribution As with both SEC-DED and Hamming codes, the effect of using a coded data stream has very little effect on the distribution of the number of errors per block. This is somewhat more surprising, since BCH codes use a considerably larger proportion of each block for parity bits, especially for smaller block sizes. It would therefore be reasonable to expect that some effect on the distribution of the number of errors per block would be noticed. However, since this is not the case, assuming the results are accurate, the effects of BCH codes may be predicted using the uncoded data stream with reasonable accuracy.

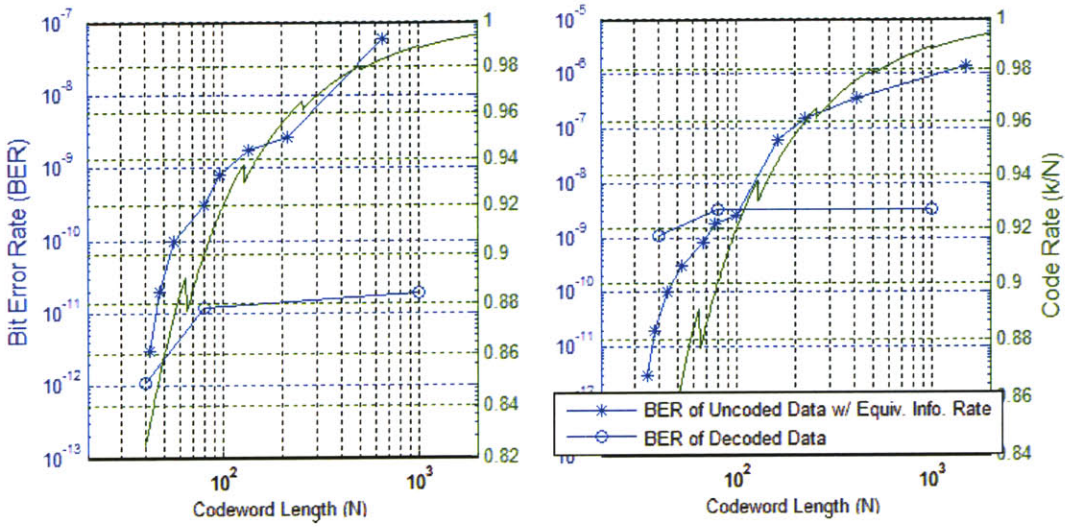

Figure 27: Effectiveness of SEC-DED Codes with Block Size of 40, 80, and 1000. The left graph shows results for 6 Gb/s, while the right graph is for 6.25 Gb/s. The raw bit error rates are approximately 2×10^{-7} and 4×10^{-6}, respectively. The BER for block sizes of 40 and 80 are based on actual correction circuits, while the BER for $N = 1000$ is an approximation of the actual BER. All detection is assumed to be eventually corrected accurately.

Results of Error Correction using BCH Codes The results of using BCH codes for 2-error-correction is shown in Figure 28. All correction is approximated, because shortened BCH codes are not particularly suitable for FPGA implementation. The raw performance of t=2 BCH codes is similar to that of SEC-DED codes, described in the previous section, since they both are designed to handle 2-bit errors. The main difference, however, is that BCH codes require many more parity bits. To correct two errors, the BCH code must have double the number of parity bits of Hamming codes. The code rate would be even lower if more powerful BCH codes are used. SEC-DED codes, on the other hand, need only one parity bit more than Hamming codes. The effect of the extra parity bits is reflected in the graphs of Figure 28, where the curve representing the uncoded data is shifted much further to the right compared to Figure 27.

The graphs show that a block size of approximately 80 is required at 6 Gb/s for the performance of the BCH code to be as reliable as the uncoded data. At 6.25 Gb/s, the same point is even higher, at around a codeword length of 200. Therefore, at small block sizes, BCH codes have either a negative effect or marginal benefit over uncoded data at the same information rate. The performance of using a larger

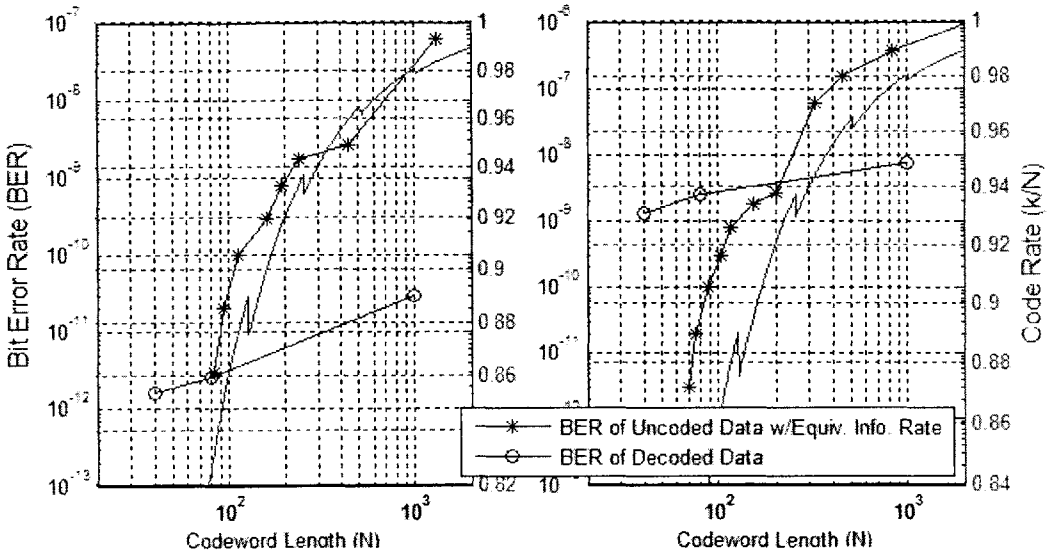

Figure 28: Effectiveness of t=2 BCH Codes with Block Size of 40, 80, and 1000. The left graph shows results for 6 Gb/s, while the right graph is for 6.25 Gb/s. The raw bit error rates are approximately 2×10^{-7} and 4×10^{-6}, respectively. No undetected errors were observed in testing any of these codes.

block size degrades slowly compared to the corresponding uncoded data performance, from 2×10^{-12} to 4×10^{-11} at 6 Gb/s and 10^{-9} to 8×10^{-9} at 6.25 Gb/s. Thus, only at very high block sizes, where the slope of the rate curve starts to flatten, can significant improvement be observed. For block sizes of 1000, there is an improvement of three orders of magnitude at 6 Gb/s and 1.5 orders of magnitude at 6.25 Gb/s.

Results of Error Detection using BCH Codes The Hamming distance of a t=2 BCH code is at least five. This means that the three codes implemented are all capable of detecting every error pattern of less than weight five. The codes are also able to detect most other error patterns of greater weight. At 6 Gb/s, a test of 8.6×10^{12} 40-bit codewords produced only one error greater than weight four, which was successfully detected. Similarly, no errors went undetected for block sizes of 80 and 1000 for 49 total high-weight error patterns. Thus, the BER after error detection at 6 Gb/s is likely less than 10^{-13}.

At 6.25 Gb/s, no errors went undetected in 167 opportunities for $N = 1000$, 227 opportunities for $N = 80$, and 38 opportunities for $N = 40$. Since more than 10^{12} bits were sent for each test, the resulting BER using detection at 6.25 Gb/s is at most 10^{-11}.

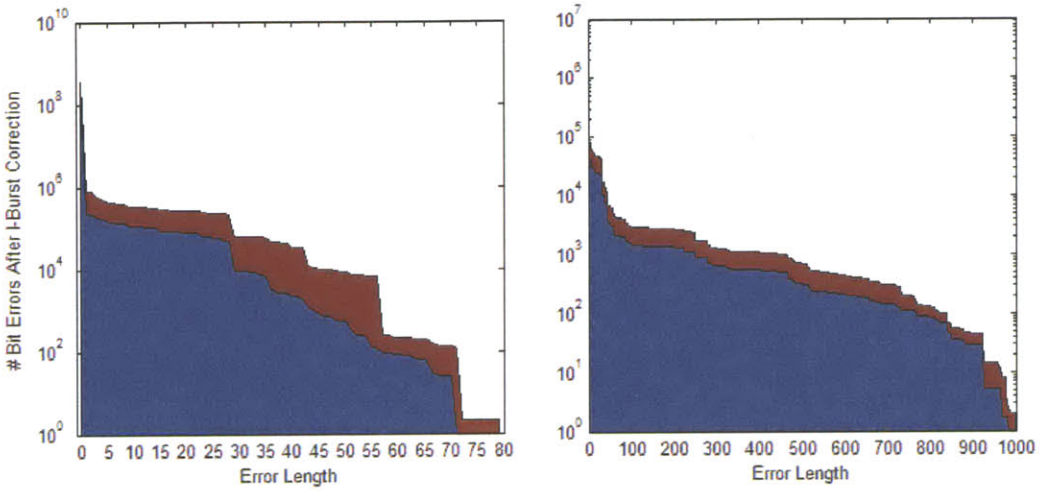

Figure 29: Data Dependency of Coded Fire Data Stream at 6 Gb/s with $N = 80$ (left) and $N = 1000$ (right). The area curve shown in blue (in front) is for the uncoded data stream, while the curve in red is the $l = 4$ Fire coded data stream. The graphs are essentially cumulative distribution functions which show how many total bit errors remain after $l-$burst-error-correction. Clearly, the number of errors remaining decreases as the strength of the code increases. The tests for $N = 80$ are normalized to 10^{13} bits and 10^{14} bits for $N = 1000$

The actual bit error rates cannot be determined without waiting for an impractically long time to generate enough errors for an accurate estimate, and thus the bit error rates using BCH codes may in fact be much lower than the upper bounds given in this section.

6.4.5 Results for Fire Codes

Effect of Redundancy on Bit Stream's Error Distribution For the previous codes, the number of bit errors per word of the coded data is compared to that of an uncoded stream. Fire codes, however, correct based on the length of the errors rather than the number of errors. Figure 29 shows the number of total bit errors remaining after l-burst-error-correction is used. The results are normalized to 10^{13} bits from 1.7×10^{14} bits and 7.2×10^{12} bits for $N = 80$. For $N = 1000$, the tests are 1.2×10^{14} bits long and 3.6×10^{14} bits long, respectively, and normalized to 10^{14} bits. The coded data has fewer total errors at each error length, so the overall bit error rate for the uncoded data is lower no matter how powerful of a Fire code is chosen. The difference is generally far less than an order of magnitude, so the difference does not affect the resulting BER in any significant way, nor does it necessarily imply that the difference is attributed to coding. Other than the height difference of the area curves, the actual distribution of

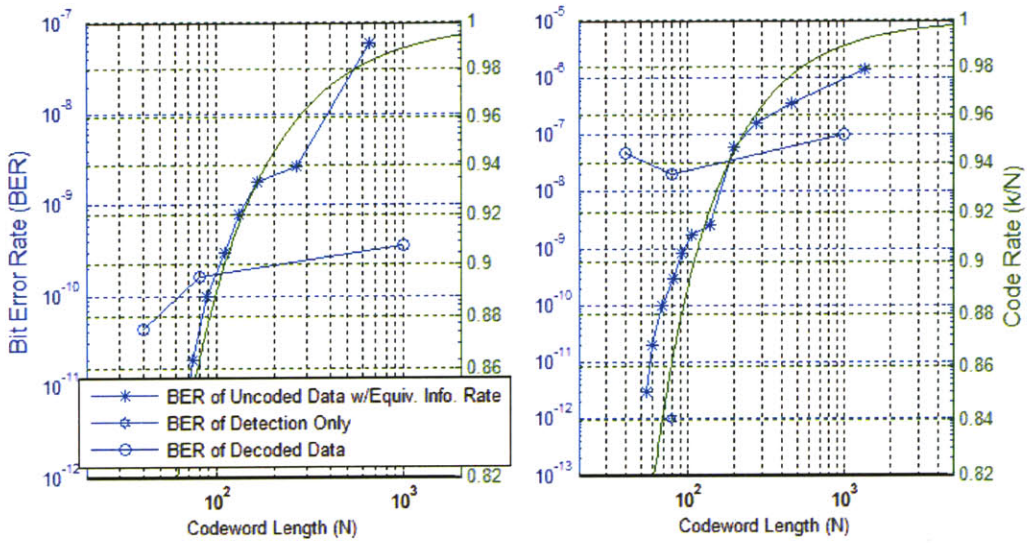

Figure 30: Effectiveness of Fire Codes with Block Size of 40, 80, and 1000. The left graph shows results for 6 Gb/s, while the right graph is for 6.25 Gb/s. The raw bit error rates are approximately 2×10^{-7} and 4×10^{-6}, respectively. Only one test, that of the (80,69) Fire code at 6.25 Gb/s, produced any undetected errors.

lengths s very similar, especially for the (1000,985) Fire code. In other words, the coded area curve is just the uncoded curve shifted up. Thus, even if the exact BER cannot be predicted by the uncoded data, the improvement expected from choosing a more powerful code can be predicted. For example, at 6 Gb/s, for both uncoded data and coded data, it is clear the BER for an $l = 30$ Fire code is a whole order of magnitude better than a $l = 29$ Fire code.

Results of Error Correction Using Fire Codes Figure 30 shows the results from using an l=4 burst-error-correcting Fire code at 6 Gb/s and 6.25 Gb/s. Notice that, unlike all previous codes, the curve indicating the code rate is not jagged. This is the curve for the maximum code rate for a Fire code, which depends on the burst-error-correcting ability rather than the block size. For the $l = 4$, the fewest number of parity bits possible is 11, and this is the number for which the curve is derived. For the codes implemented, however, the actual number of parity bits for block sizes of 40, 80, and 1000 are 13, 11, and 15, respectively.

For the three block sizes implemented, the approximate BER results are all well within an order of magnitude. The BER at 6 Gb/s is approximately 10^{-10}, and the BER at 6.25 Gb/s is between 10^{-8} and

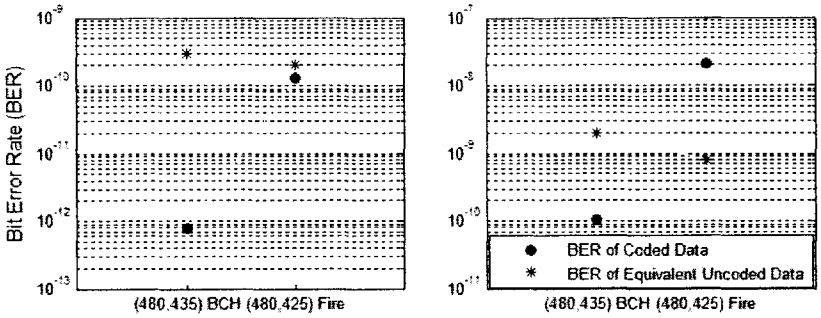

Figure 31: Effectiveness of correction using the $t = 5$ (480,435) BCH and the $l = 18$ (480,425) Fire code at at 6 Gb/s (left) and 6.25 Gb/s (right). Detection with these codes yielded no undetected errors in $> 10^{14}$ bits at 6 Gb/s and 10^{12} bits at 6.25.

10^{-7}. Thus, increasing the block size is an effective way of improving the performance, especially since the number of parity bits does not necessarily increase with block size. For very large block sizes, the expected BER improvement is almost three orders of magnitude at 6 Gb/s and one order of magnitude at 6.25 Gb/s.

Results of Error Detection Using Fire Codes The $l = 4$ Fire codes implemented in the project are capable of detecting all bursts of length eight, and most errors of greater than eight. The discussion of error detection using the t=2 BCH codes generally applies here, except there are far more instances of received codewords with large error lengths than there are codewords with many bit errors. Tests of approximately equal number of bits were run for the Fire codes as with the BCH codes, and there was only one test that had any undetected errors. Of 10^{11} codewords, the (80,69) shortened Fire code let one 6-bit error of length 43 pass undetected at 6.25 Gb/s.

Thus, the resulting BER using detection only is less than 10^{13} for a data rate of 6 Gb/s. The (80,69) Fire code produces a BER of approximately 10^{-12}, while the bit error rate of the link when using either the (40,27) or the (1000,985) Fire code is less than 10^{-11}.

6.4.6 Results for More Powerful BCH and Fire Codes

Results of Error Correction The results of using the $t = 5$ (480,435) BCH code and the $l = 18$ (480,425) Fire code are shown in Figure 31. As expected, the BCH code performs significantly better

than the Fire code at both data rates. For the BCH code, although the order of magnitude BER improvement is not greater than the $t = 2$ BCH code, the more powerful code is necessary in order to produce results closer to the 10^{-15} bit error rate goal. The BER of the more powerful code at $N = 480$ is less than 10^{-12} while the BER of the $t = 2$ code is approximately 10^{-11} at 6 Gb/s. At 6.25 Gb/s, these BERs are nearly 10^{-8} versus approximately 10^{-10}. Compared to a BSC, the improvement from using a much more powerful, hardware-consuming, and energy-consuming code is quite disappointing.

The results of using the Fire code is even more disappointing. At 6.25 Gb/s, the gains achieved by the code are not even sufficient to overcome the code overhead. Furthermore, compared to the $l = 4$ code, the more powerful Fire code brings the BER down further from 3×10^{-10} to 10^{-10} at 6 Gb/s and from 8×10^{-8} to 2×10^{-10} at 6.25 Gb/s. The poor performance stems from the fact that the $l = 4$ code is able to correct most of the short-length errors, so increasing the burst-correcting ability does very little to the overall performance.

Results of Error Detection As previously stated, these two codes are considerably more powerful than the other codes implemented. Thus, it is not surprising that the tests did not yield any undetected errors in more than 10^{14} bits at 6 Gb/s and 10^{12} bits at 6.25 Gb/s.

6.5 Code Comparison

The four codes implemented in the project produce vastly different results. At 6 Gb/s, the BER improvement over uncoded data with the same throughput for each code and block size is illustrated in Figure 32. The equivalent graph for 6.25 Gb/s is shown in Figure 33.

From Figures 32 and 33, three main conclusions can be drawn. The first conclusion is that clearly detection outperforms correction. The Hamming code in detection mode, the SEC-DED code, and the t=2 BCH code can be used to implement the three different types of 2-error-correcting control systems. A Hamming code used as a CRC can be implemented in an ARQ system. The SEC-DED code is meant to be used in a hybrid FEC-ARQ scheme, and the BCH code can be used in a FEC system. From the two figures, the Hamming code outperforms the SEC-DED code by two or three orders of magnitude, and the SEC-DED code outperforms the BCH code by an order of magnitude for some block sizes. Recall that the

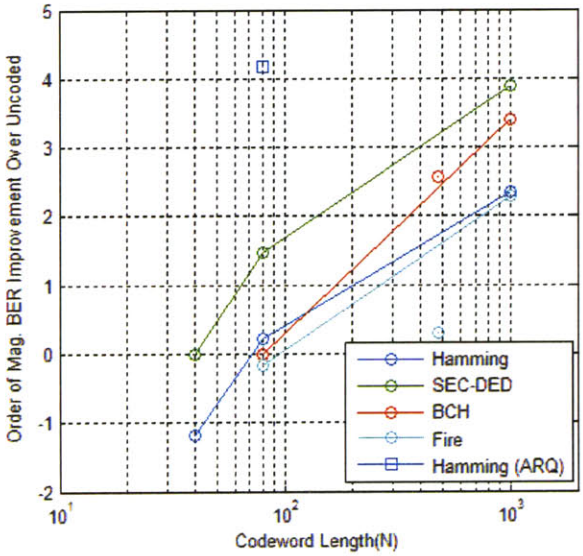

Figure 32: Bit Error Rate Improvement For Each Code at 6 Gb/s. The BER improvements using ARQ with more powerful codes than Hamming are not plotted, because they are off the charts.

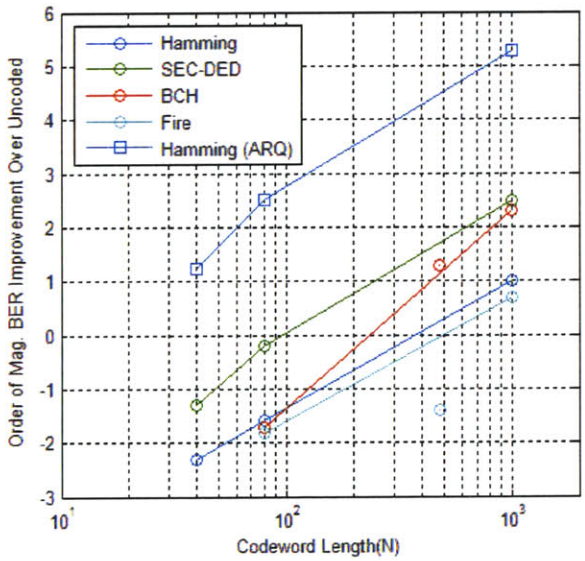

Figure 33: Bit Error Rate Improvement For Each Code at 6.25 Gb/s. The BER improvements using ARQ with more powerful codes than Hamming are not plotted, because they are off the charts.

SEC-DED code does not perform up to the code's detection capacity, which would require a much more complex detection circuit. Thus, it is likely that at its best, the SEC-DED code would be vastly superior to the BCH code. The Hamming code outperforms the other two codes for two reasons. First, detection can be implemented at a higher code rate, since it requires less redundancy. Also, detection is capable of handling a large number of error patterns outside its guaranteed detection range, while correction is often forced to miscorrect the errors. SEC-DED, which utilizes detection to a lesser degree, performs better than BCH for these same reasons.

A second conclusion that is observed is the significant increase in performance when the block size is increased. This phenomenon is mentioned in the discussion for each code, but the significance is worth reiterating. The code rate for a linear block code generally increases with block size. Thus, as long as a large enough codeword length is used, a very powerful code can be implemented at a high enough rate to overcome the overhead. For example, the code rate for a t=5 BCH code would be far too low if a small block size were used (i.e. a (63,36) BCH code has $R = 0.57$), but from Figures 32 and 33, it is clear that the code can achieve significant gains for $N = 480$. Similarly, another observation that can be drawn from the figures is that at a high enough block size, the difference between the various code rates becomes negligible. The most clear illustration of this fact can be seen by comparing the curves for the SEC-DED and BCH codes at either data rate. Both codes achieve virtually the same BER, since both BERs are dominated by triple-bit errors. Thus, the performance improvement stems almost completely from the code rate. As the block size increases, the difference between the two curves shrinks so that at $N = 1000$, the curves are separated by far less than an order of magnitude, especially at 6.25 Gb/s.

Finally, length-based correction (i.e. a Fire code) does not perform nearly as well as correction based on the number of errors (i.e. a Hamming, BCH, or SEC-DED code). Fire codes are not as efficient as the best burst-error-correcting codes, but the performance of the code does not merit examining more efficient codes. Fire codes are not even able to perform better than Hamming codes at either data rate. Increasing the burst-error-correcting capability only decreases the BER improvement, as can be seen by the circle representing the (480,425) $l = 18$ Fire code. This result is not surprising. Many of the error lengths are short when there are only two or three errors in the word, but as the number of errors per

codeword increases, so does the error length (see Figure 21). Thus, it is far more worthwhile to focus on correcting all the errors of lesser weight rather than only the ones of short length.

6.6 Remarks on Hardware Requirements of Codes

The complexity of implementing a code in hardware varies based on what code is used, the size of the codeword, and the decision between correction, detection, and simultaneous correction and detection. Current transceivers are implemented such that if coding were used, the parity bits would be formed in parallel at the transmitter and decoded serially at the receiver. It is difficult to estimate the complexity of the transmitter based on the type of code alone, because the complexity depends partially on the number of 1's in the matrix of the generator matrix. Thus, of the codes studied in the project, only SEC-DED codes guarantee the simplest encoding circuit. Other than that, clearly the more parity bits required by the code, the more registers and total xors are required to generate the parity bits.

At the decoder, a cyclic or shortened cyclic code is advantageous, especially if detection is used. With detection, only a simple LFSR of size m is needed to form the syndrome. The syndrome is enough to determine whether retransmission is necessary. The hardware of the circuit then grows linearly with the number of parity bits needed, and the block size has little bearing on the hardware of the circuit. On the other hand, if correction is used, then besides the syndrome circuit, a correction circuit and a buffer of size n is necessary to perform the correction properly. Thus, a correction circuit requires many more registers and additional hardware. It is evident, therefore, that detection is promising for reasons other than the performance benefits shown in previous sections. Also, since the block size does not greatly affect the detection circuit, increasing the block size may prove to be a viable option in an ARQ system.

6.7 Alternative Codes that May Yield Good Results

One of the most interesting results of the project is the performance of the Hamming code to detect errors. At 6 Gb/s, the Hamming code is only able to obtain a BER of 3×10^{-10} from a raw bit error rate of 2×10^{-7}. With detection, however, the bit error rate is reduced all the way down to 3×10^{-14}, a additional four orders of magnitude. Similarly, at 6.25 Gb/s the BER drops from around a raw bit error

rate of 10^{-6} to 10^{-11}. It may be that a code with a Hamming distance of four instead of will be enough to reduce the bit error rate all the way down to the desired mark of 10^{-15}. Perhaps the SEC-DED code can be used, though the fact that it is not cyclic deters from its appeal. According to [12], a CRC for a (63,56) code generated by $\mathbf{g}(X) = X^7 + X^5 + X^4 + X^2 + X + 1$ is particularly effective for detection over a BSC. This code may be worth studying, because it has a high code rate of 0.89 and a small enough block size to be implemented using current technology. If a higher code rate is needed, [12] lists CRCs with data lengths up to 2048 bits.

If the retransmission of worst-case patterns turns out to be problematic for the throughput of an ARQ system, then it may be desirable to implement a hybrid FEC-ARQ system. Then, a worst-case pattern will be accepted correctly when one error, or more depending on the code, is received within the codeword. Although the SEC-DED code performs well, it may be necessary to implement it in such a way as to be able to detect all odd-weight error patterns that do not produce the same syndrome values as single-bit errors. Even with the improvement, single error correction and double error detection may not be enough to achieve the desired results. Then, the Hamming distance should be increased to be able to detect even more errors and possibly correct more errors.

7 Conclusion

The work described in this document provides a number of useful insights into the effectiveness of using codes for links. First, the distribution of both the error lengths and the error weights for blocks of various lengths is discovered. The distribution of error weights shows that although the errors are correlated, occurrences of error patterns with weights of more than eight are rare. Furthermore, the error length distribution shows that the expected error length grows rapidly with the weight of the error pattern, indicating that error length-based correction may not be particularly effective.

Since the BER is low and error weights typically are not large, simple codes are implemented to test the performance of existing codes. These codes are Hamming, BCH, SEC-DED, and Fire codes, of which the first three are random-error-correcting codes. Some of these codes are implemented for detection, correction, simultaneous detection and correction, or for more than one of the above. Overall,

the results confirm that purely random-error-correcting codes perform better than purely burst-error-correcting codes. Furthermore, error detection proves to be a far superior option for links, since a back-channel exists, and detection requires less hardware and has higher BER gains. However, to fully analyze the results of using an error detecting code, a full ARQ scheme must be implemented. This necessity is described further in the following section.

7.1 Recommendation for Future Work

Since error patterns with weight greater than eight occur very rarely, a random-error-detecting code can be used to obtain low bit error rates. The other option would be to use error correction. Two possible BCH codes that can correct eight errors are the (255,191) and the (1023,943) codes, which have rates of 0.75 and 0.92. These codes have low rate considering the block size. Thus, error correction has both low throughput and would require a large amount of hardware. In comparison, error detection can use a code that has a Hamming distance of 9, like a $t = 4$ BCH code. The equivalent BCH codes are the (255,215) and the (1023,991) codes, which have much higher rates of 0.84 and 0.97 and less than half the hardware.

The full performance of an error-detecting code within an ARQ system is still unknown, since the approximations made in this project do not account for some factors that may impact the performance of the ARQ system as a whole. The two factors that are potentially problematic are the number of retransmissions necessary for a worst-case pattern to be accepted, and the frequency of detected errors that end up being undetected errors in a retransmission. Thus, it is necessary for the codes implemented in this project to actually be used in an ARQ system. If the results are similar to those described in this document, then the recommended codes addressed in the previous section should be tested.

There are even more factors to consider when implementing the ARQ protocol that are not considered in the project. One such concern is choosing an appropriate block size. From a coding standpoint, based on the results of this work, it is wise to choose large block sizes. However, from a retransmission standpoint, smaller block sizes are desirable so that both the probability of retransmission is lower and the number of bits to retransmit is fewer. In fact, for a BSC, the block size that achieves the best throughput is a block of size one [21].

Finally, another concern that must be addressed is the complexity of implementing a retransmission scheme, and whether that uses less energy than implementing a more powerful error correcting code. The simplicity of detection circuits over correction circuits is counteracted by the added hardware required by an ARQ system, such as a transmitter and possibly a receiver buffer. There are many types of ARQ systems, and these must be explored, along with a given code, to determine the best system based on achievable throughput and complexity.

From the findings of this work, detection is the route with the most potential for link coding. To complete the study, the implementation and examination of appropriate ARQ systems is necessary to determine whether detection, and coding in general, is suitable for links. Afterward, if coding continues to be a promising solution, then the further steps detailed in the Problem Statement should be taken.

7.2 Final Recommendation

Due to the promising results of both error detection, which produces a BER improvement of approximately five orders of magnitude at 6.25 Gb/s and $HD = 3$, and simultaneous error detection and correction, which produces a BER improvement of 2.5 orders of magnitudes at 6.25 Gb/s and $HD = 4$, there is potential for coding to be effective at combating link errors. The low BER requirement of 10^{-15} and the desire for high energy-efficiency essentially rules out the possibility of using an FEC scheme, especially considering the results of the poor results of the $t = 5$ BCH code and the $l = 18$ Fire code. However, there are still many avenues to explore with retransmission schemes that can and should be looked into in the near future.

To summarize, the answers to the questions posed at the beginning of the document are:

- *What codes would work better than Hamming? And more significantly, what properties do these codes possess that make them so appropriate for the link environment?* Hamming is an effective code for links, since single-bit errors dominate the bit error rate. However, especially since the channel is not BSC, double-errors, triple-errors, etc. are too common for such a low-power code to produce the necessary results. It appears that it is not possible for an error control scheme to achieve the desired results without being able to correct at least all patterns with eight bit errors,

whether that system includes a t=8 error-correcting code or a code with a Hamming distance of nine to detect all 8-weight errors.

- *Given a set of criteria on block size, data rate, etc., what is the best code to use? What makes that code better than all of the others?* The results of the code do not produce definite answers to these questions, but can comment on trends. Increasing the block size increases the performance of the code, since the code rate increases and the number of bit errors per word does not increase. Furthermore, at large block sizes, most codes have virtually the same code rate, so extremely powerful codes may be chosen as long as the corresponding hardware can be implemented. As for the data rate, comparing the general results at 6 Gb/s and 6.25 Gb/s, it is easier to improve upon a BER that is lower. Thus, in the future it will be necessary to determine how powerful to make the equalizers versus how powerful to make the code.

- *Which type of error control scheme has the most potential to work for links?* From the results, detection is shown to be far superior to correction. The true effectiveness of a complete ARQ system is still unknown, but its potential advantages over FEC systems are clear.

- *Is there a single combination of variables (i.e. block size, data rate, code) that is clearly superior? If not, then what are the most promising results that should be looked into further in the future?* The results of using detection are extremely promising, though the best code and block size has not been determined. Based on the results generated, the suggested codes may in fact produce low enough bit error rates and hardware complexity.

References

[1] R. Kollipara, "Design, modeling, and chracterization of high-speed backplane interconnects," *DesignCon*, 2003.

[2] V. Stojanovic and M. Horowitz, "Modeling and analysis of high-speed links," *Custom Integreted Circuits Conference Digest of Papers*, 2003.

[3] J. Zerbe, "Design, equalization and clock recovery for a 2.5-10Gb/s 2-PAM/4-PAM backplane transceiver cell," *IEEE International Solid-State Circuits Conference*, february 2003.

[4] V. Stojanovic and J. Zerbe, "A systems approach to building modern high-speed links," *IWSOC*, 2005.

[5] V. Stojanovic, A. Amirkhany, and M. Horowitz, "Optimal linear precoding with theoretical and practical data rates in high-speed serial-link backplane communication," *IEEE International Conference on Communications*, 2004.

[6] A. Amirkhany, V. Stojanovic, and M. Horowitz, "Multi-tone signaling for high-speed backplane electrical links," *IEEE Global Communications Conference*, 2004.

[7] S. Lin and D. J. Costello, *Error Control Coding*, 2nd ed. Upper Saddle River, New Jersey: Pearson Prentice Hall, 2004.

[8] A. Hocquenghem, "Codes corecteurs d'erreurs," *Chiffres*, vol. 2, pp. 147–156, 1959.

[9] R. Bose and D. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Information Control*, vol. 3, pp. 68–79, March 1960.

[10] R. Hamming, "Error Detecting and Error Correcting Codes," *Bell System Tech. J.*, 1950.

[11] M. Y. Hsiao, "A class of optimal minimum odd-weight-column SEC-DED codes," 1970.

[12] P. Koopman and T. Chakravarty, "Cyclic redundancy code (CRC) polynomial selection for embedded networks," *The International Conference on Dependable Systems and Networks*, 2004.

[13] L. Thon and H.-J. Liaw, "Error-correction coding for 10 Gb/s backplane transmission," *DesignCon*, 2004.

[14] D. Carney and E. Chandler, "Error-correction coding in a serial digital multi-gigabit communication system: Implementation and results," *DesignCon*, 2006.

[15] A. Bessios, W. Stonecypher, A. Agarwal, and J. Zerbe, "Transition-limiting codes for 4-PAM signaling in high speed serial links," in *IEEE Global Telecommunications Conference*, December 2003, pp. 3747–3751.

[16] A. Ho, V. Stojanovic, F. Chen, C. Werner, G. Tsang, E. Alon, R. Kollipara, J. Zerbe, and M. Horowitz, "Common-mode backchannel signaling system for differential high-speed links," in *Symposium on VLSI Circuits. Digest of Technical Papers.*, June 2004, pp. 352–355.

[17] Xilinx. (2004) Ug035: Rocketio transceiver x user guide. [Online]. Available: http://www.xilinx.com/bvdocs/userguides/ug035.pdf

[18] ——. (2004) Xapp661: Rocketio transceiver bit error rate tester. [Online]. Available: http://www.xilinx.com/bvdocs/appnotes/xapp661.pdf

[19] "General requirements for instrumentation for performance measurements on digital transmission equipment," *ITU-T Recommendation O.150*, May 1996.

[20] P. P. Chu and R. E. Jones, "Design techniques of FPGA based random number generator," in *Military and Aerospace Applications of Programmable Devices and Technologies Conference*, September 1999.

[21] J. Morris, "Optimal blocklengths for ARQ error control schemes," *IEEE Transactions on Communications*, vol. COM-26, pp. 488–493, January 1979.